

bea CD INSIDE!

CAREER OPPORTUNITIES SECTION

JAVA DEVELOPER'S JOURNAL

The World's Leading Java Resource

June 2000 Volume: 5 Issue: 6

JAVA DEVELOPERS JOURNAL.COM

Announcing... **XML DevCon 2000**
 Coming June 25-28, 2000

JavaCON 2000 September 24-27, 2000

From the Editor
by Sean Rhody pg. 7

Guest Editorials
 by Richard Soley pg. 8
 by Jeremy Allaire pg. 10
 by George Paolini pg. 12
 by Alex Roedling pg. 14

Straight Talking
by Alan Williamson pg. 30

E-Java
by Ajit Sagar pg. 48

Programming Techniques
by Gene Callahan & Rob Dodson pg. 92

EJB Home
by Jason Westra pg. 118

Corba Corner
by Todd Scallan pg. 138

Interview
Alan Armstrong of KL Group pg. 160

IMHO
by Bruce Scott pg. 210



BUILDING **E-COMMERCE** APPLICATIONS USING BUSINESS COMPONENTS FOR **JAVA**

INSIDE FREE CD! WEBLOGIC SERVER RELEASE 5.1

CAREER OPPORTUNITIES PAGE 177 **HOT JAVA JOBS** SECTION

Jim Milbery 16

Java & Robotics: How to Control a Robot Over the Internet *Simple robotics with java servlets*  Darrel Riekhof Keith Fligg **34**

Feature: SQL Embedded in Java PART 2  *Continuing our miniseries on SQLJ* Ekkehard Rohwedder **42**

Java Utilities: A Java File Search Utility  *It's functional as it stands - and you can add to it* Pat Paternostro **54**

Feature: Using the JMS with EJBs  *Bring these two powerful enterprise technologies together* Scott Grant **58**

Feature: Applying Patterns to JDBC Development  *A real-world database scenario* Christian Thilmany **80**

Building EJBs with VisualAge for Java  *Eliminate the frustration of developing EJBs by hand* Lucy Barnhill, Angus McIntyre & Rob Stevenson **98**

Feature: Self-Contained Client Applets Using Swing  *Create a GUI outside the browser's frame* Thomas Czernik Rolf Kamp **108**

Feature: Bean-Managed Persistence Using a Proxy List  *Saving dependent objects in a relational database* Daniel O'Connor **126**

Feature: Extending Your Applications with BSF  *BSF provides a universal scripting platform for Java* Rick Hightower **148**

Progress Software

www.sonicmq.com/ad1.htm

Protoview

www.protoview.com

Bluestone Software

www.bluestone.com

JAVA DEVELOPER'S JOURNAL

CONTENTS

VOLUME: 5 ISSUE: 6 JUNE 2000

COVER STORY

Building E-Commerce Applications

Construct a complete application with a JSP



SQL Embedded in Java: Mixing Java & SQL: Part 2

SQLJ, the standard for embedding database SQL statements

FEATURE

FEATURE

Using the JMS with EJBs

Bring these two powerful enterprise technologies together



Applying Patterns to JDBC Development

A real-world database scenario

FEATURE

FEATURE

Self-Contained Client Applets Using Swing

Create a GUI outside the browser's frame



Bean-Managed Persistence Using a Proxy List

Saving dependent objects in a relational database

FEATURE

FEATURE

Extending Your Applications with BSF

BSF provides a universal scripting platform for Java



FROM THE EDITOR

sean rhody 7

GUEST EDITORIALS

richard soley 8
jeremy allaire 10
george paolini 12
alex roedling 14

STRAIGHT TALKING

alan williamson 30

JAVA & ROBOTICS

darrel riekhof & keith fligg 34

E-JAVA

ajit sagar 48

JAVA UTILITIES

by pat paternostro 54

JAVA & ORACLE

by samir shah 74

PROGRAMMING TECHNIQUES

by gene callahan
& rob dodson 92

VISUALAGE REPOSITORY

by lucy s. barnhill, angus
mcintyre & rob stevenson 98

EJB HOME

by jason westra 118

CORBA CORNER

by todd scallan 138

JDJ NEWS

158

INTERVIEW

with alan armstrong
of KL Group 160

IMHO

by bruce scott 210

No
Magic
www.magic-draw.com

TogetherSoft Corporation

www.togethersoft.com

Today's Silver Bullet Is Tomorrow's Legacy

Addressing the legacy integration conundrum

WRITTEN BY RICHARD SOLEY



One of the most delightful parts of my job is to travel the world, sharing the Object Management Group's vision of integrated, interoperable systems with varying sizes of audience – from as few as 10 people to as many as 10,000 – in every corner of the planet.

While the travel can sometimes be grueling, it's worth it when I get a question or two after a speech that shows that someone has experienced the epiphany I myself had in 1989, when I realized that no business can be automated by software systems until the individual applications that automate individual processes are integrated.

There are a few favorite questions that I ask at these events in order to evoke audience participation. My favorite by far is, "What's the definition of a legacy system?" Unfortunately audiences are getting smarter – or perhaps they've just heard me speak before! But until recently most of the answers were in the set:

- "Something written in COBOL (or Jovial or PL/I)"
- "Something that runs on an IBM 370 (or IBM 360 or Burroughs machine)"
- "Something that nobody knows how to use anymore"
- "Something for which the source code has been lost"
- "Something written by programmers long dead and forgotten"

I certainly understand those last two. I once had to work with a million-line FORTRAN program, the source code for which had been lost and the programmers of which had passed away! My own definition, however, is much simpler: a legacy system is a system that runs.

Whether it was written in IBM Assembler F in 1963 and is currently being run in a 1401 emulator on a 360 emulator on a 370 emulator on a Hitachi mainframe or written yesterday morning in Java, if it is deployed now, it is a legacy system. Any future software development that we must do in the enterprise must either integrate with that legacy, or replace it – and we all know that software doesn't die, so we are left with only one choice: integration.

I don't think anybody is surprised so far – certainly, as the chairman of a consortium focused entirely on standards for integrating systems, I am not surprised. Instead, the continuing surprise for me is the way our industry responds to this legacy integration conundrum. That response has been, historically, and apparently will continue to be in the future, the continuous and uninterrupted introduction of silver bullets designed to slay vampires (complex application integration problems).

Since the OMG was founded, these bullets have included C++, Java, Enterprise JavaBeans, OLE/ActiveX/COM/DCOM/COM+/DNA and now XML. (I have bad news for you: XML isn't the universal data interchange format; it's yet another universal data interchange format.) Each of these previous silver bullets has been another legacy to deal with when the next bullet whizzed by, and this will continue to be true.

Oh yes, another technology touted as universal, a source of homogeneity to cast out the heterogeneity of the world is...CORBA, from the Object Management Group. What makes us any different?

The difference is that CORBA is not itself homogeneous. The OMG has always focused on bridging diverse systems and CORBA has changed in dramatic ways over the years to fulfill that vision. CORBA now includes specifications to integrate directly with such dissimilar languages as C, C++, Java, COBOL, Ada, Smalltalk and Lisp; to integrate seamlessly with Microsoft COM and the Java Platform; and to seamlessly compose components with Enterprise JavaBeans and Microsoft MTS components.

Furthermore, as the industry has moved to more structured development techniques and practices, OMG has moved with it. The OMG's Unified Modeling Language is a universally supported meta-model and notation that has unified the industry around a single standard, a standard that includes (among other things) support for metadata repositories, XML-based repository integration, common data warehouse standards for database integration, and mappings to CORBA and other technologies. Again, this allows integration based on the legacy of not only programming artifacts (code), but even abstract designs.

Furthermore, the OMG has moved rapidly in the past three years to build on these infrastructure successes and the open, neutral OMG consensus process by publishing standards in medical systems interoperability, standardized workflow processing, product data management integration, telecommunications systems convergence and many other areas. Did you know that OMG has published standards for air traffic control? How about for human genome research?

All of this activity is structured around the unceasing need for integration. The current spate of enterprise application integration solutions – another new magic bullet – finds us in the same situation when Company A (which had used EAI tool X) merges with Company B (which had used EAI tool Y). Suddenly it's the integration tools themselves that need integrating!

The search for silver bullets in our industry will never cease. Perhaps it shouldn't – maybe someday we'll really find one. In the meantime, after the silver bullet hits a brick wall, somebody has to pick it up and integrate it with the other legacies. Eight hundred companies now achieve that necessary interoperability through the open, neutral OMG process – we'd love to have you participate! 🍌



AUTHOR BIO

Richard Soley is chairman and CEO of the Object Management Group.

WebGain

www.webgain.com

Beyond Java: The Metamorphosis of an Operating Platform

WRITTEN BY JEREMY ALLAIRE



By most people's estimate, it's the fifth anniversary of Java. Five years ago, with Netscape in tow, Sun unveiled Java, declaring that the Java programming language would be the next Web revolution. At the time HotJava was the "killer app" for Java; more a proof of concept than a competitive browser platform, it demonstrated that there could be more to the Web than plain old HTML.

Within a year all the major browser platforms included a 1.0 version of the Java Runtime and applets were considered the next big thing after plug-ins. Both, it turned out, were pretty much useless for e-commerce and interactive Web sites: all the action was on the server. Java was declared dead by many industry pundits and its role on the client side of the Internet equation diminished massively.

Sun and other vendors began accordingly to shift their attention server-side, since computing application servers were becoming the new operating platforms for the Internet. After three years this effort has emerged as J2EE, arguably one of the fastest-growing enterprise computing architectures in history.

Why is all of this so important? It matters because today, while Java is on the threshold of being established as a new-breed operating platform for the Internet, many customers and developers still understand it, narrowly, as an Internet programming language.

With the metamorphosis of Java from cool Internet programming environment to foundation platform for Internet applications comes a need to rearticulate the role of Java the platform versus Java the language. As Java becomes more strategic to the industry overall, the transformation also raises fundamental questions for customers and vendors about the openness of the platform.

If Java is to the Internet environment what Windows was to the PC environment, and if by definition foundation Internet technologies require open architectures, then we're faced with a long-term ques-

tion about the overall openness of Java. In the Internet environment, there are really four major categories of technology, each with their own degree of openness:

1. **Open standards-based technologies:** The vast majority of technologies running the Internet are based on open standards created by vendor-driven standards bodies such as the IETF, W3C, ISO and ECMA. These include TCP/IP, HTTP, HTML, XML, RARP, X.509 and hundreds of other core technologies. These standards still rely on vendor or community implementations.
2. **Open technologies defined by process and license:** A vast amount of Internet technology in place today is literally and truly open in legal terms, typically as defined by the General Public License (GPL) or a BSD-style license. These technologies include operating systems (Linux), Web servers (Apache) and application development and runtime technologies (Perl, PHP, Python, C/C++, and so on).
3. **Open technologies defined by process, but not license:** This includes vendor-controlled technologies but with an open community process in the evolution of those technologies. Great examples of this include Java or an older example such as ODBC, driven by Microsoft. Nonetheless, in both cases a single vendor ultimately owns and controls the technology.
4. **De facto standards with broad distribution:** In many cases, commercial success with a proprietary technology establishes a de facto standard position for a vendor technology. Examples of this include Microsoft ASP, Allaire ColdFusion, RealNetworks RealMedia and Macromedia Flash.

Today Java as a platform is an open technology by process – but not by license. With vendors and customers betting their future on this platform, what's the appropriate process, license and ultimate ownership of this critical technology? I think this will become a pressing question over the coming years.

—continued on page 28



Microsoft

msdn.microsoft.com/training

A Call to Action for the Java Technology Community

WRITTEN BY GEORGE PAOLINI, VICE PRESIDENT, JAVA COMMUNITY DEVELOPMENT, SUN MICROSYSTEMS, INC.



We live in a world of high anxiety. We're concerned about the competition, fearful we'll fall behind the curve, worried that making up lost ground might prove impossible. So we hastily turn to technology, which obligingly always seems to have a solution. Well, at least it says so in the marketing brochure....

But how much are things really changing? Are these advances really so revolutionary? Or are they simply refinements on a few good ideas?

We measure progress in the technology industry according to speed, price, weight and ease of use. (Which doesn't explain why, when I go on the road, I now lug around 20 pounds of batteries and transformers for my myriad devices and why I have no idea how to get my Palm Pilot to talk via infrared to my PC.)

Every so often, however, progress comes not in gradual increments, but in a big, earthshaking blast: Apollo 11, ARPANET, the Mac, the Web. As you might have guessed, I put Java technology in this category.

Now, as revolutionary as Java technology is, I believe that when the history books are written, the process we're using for evolving the platform will prove to be as revolutionary in its own right.

When the Java platform was first introduced in '95, it was the equivalent of Henry Ford building cars before the roads existed. An incredible piece of technology with no place to go. Fortunately, we realized early on that Java technology was bigger than Sun, and that succeeding would mean involvement of the entire industry: developers, IT professionals, software startups and industry stalwarts.

The process for working with Sun on evolving the platform was quite ad hoc and improvised at first. In many ways it had to be, given how quickly things were changing during the first few years. The industry – Sun, Java technology licensees, developers – built 80 new programming interfaces to grow Java from a language and runtime into a full-blown platform in that time.

By December 1998 we had formalized this methodology in what is now known as the *Java Community Process* (JCP). We did more than just document the process, however. We also relaxed a number of restrictions, allowing companies other than Sun to take the lead on creating new interfaces, and allowing anyone to join in these efforts.

It's a process that, while open to improvements, has worked enormously well. To date, more than 65 new interfaces have been intro-

duced. These efforts, known as *Java Specification Requests* (JSRs), are now in various phases of the six steps involved in openly evolving the Java technology.

Those steps are a Proposal, Expert Group Formation, Participant Draft, Public Draft, Final Release and Maintenance. More than 180 companies and individuals have signed up as participants in the JCP. Many have taken on leadership roles, becoming specification leads for nearly half of all JSRs. These leaders are responsible for three key deliverables: a technology specification, a reference implementation and a Compatibility Test Suite.

Specifications are detailed, written documents that outline the technical attributes of an API. The reference implementation is a working example of the specification that serves as proof of concept – proof that the technology specifications can be implemented. The Compatibility Test Suite is a collection of tests, tools and other requirements used to certify that an implementation conforms to both the applicable API specifications and the reference implementation. It helps to ensure consistent Java technology implementations across various platforms.

Is the JCP perfect? Not quite. It's clear that now, as businesses are betting not only their existence but also their success on Java technology, they are concerned that the process Sun drives is open and equitable. Fair enough. That's why we're exploring a number of changes in JCP 2.0, due to launch this summer. One change, for instance, would be the formation of an

executive committee, which would include industry stakeholders and would be responsible for approving the passage of technology specifications and determining when a technology specification is ready for public review.

There are surely many more ways to improve upon and refine the JCP, and we'll continue to work on it with your help. You can bet that any changes we make will support our No. 1 objective: to provide Java technology developers with a stable platform for rapid innovation.

I urge you to get involved in the evolution of the Java platform. Just go to java.sun.com/jcp. It's really that simple. We look forward to working with you. ☛

AUTHOR BIO

George Paolini and his team at Sun are responsible for ensuring the widespread adoption of Java, Jini and other technologies, and work with partner companies and licensees to drive the community process that evolves the technologies.

george.paolini@sun.com



WebVision

www.webvision.com

Digital Marketplace Platform: Here Today, Flexible for Tomorrow

WRITTEN BY ALEX ROEDLING

The e-commerce market opportunities in the B2B space are exceptionally greater than those in B2C. GartnerGroup, one of the leading analyst firms, predicts that B2B e-commerce will amount to over \$7 trillion by the year 2004. The interoperability and integration complexities are also multidimensional in comparison to those in B2C space. Sean Rhody, *JDJ* editor-in-chief, touched on this very topic in the March issue (Vol.5, Issue 3). Consumers have benefited from the dot-com phenomenon in terms of cost savings, convenience and automation. Buyers and sellers in the B2B space are after those same cost-saving benefits to increase bottom-line profitability.

Rise of the Digital Marketplace

Recent developments have seen the rise of the digital marketplace, a centralized location where multiple buyers and sellers meet and exchange goods, services and information. This centralized approach eases the buying and selling of products in a fragmented industry. Organizations aren't quite sure how they're going to participate in an electronic marketplace, but they know they'll have to. Primarily, they fear they'll lose their competitive advantage if they don't. But the benefits for all parties involved are tremendous.

Key Digital Marketplace Components

Digital marketplaces strive to be the business portals of the future. Their goal is to attract as many buyers and sellers as possible and provide them with the tools to exchange goods and services. Not only must they enable e-commerce, but they must also foster communication and speed the sales cycle for members to become more profitable. Marketplaces typically make their money by taking a percentage of the transaction revenue; in the future however, market makers will have to offer value-added services in order to grow their revenue and gain customer loyalty.

Marketplaces address the following functions:

- **Content:** A catalog content management system enables suppliers to offer their catalogs electronically, giving them a tremendous reach into a much wider potential audience of buyers. For buyers, a centralized location for purchasing goods allows them to find the best price, quality, terms and conditions or any other key buying criteria that may be important to them.
- **Commerce:** Although marketplaces typically service a unique trading community and provide different levels of marketplace functionality, they all share the common requirement of automating the exchange of information and transactions in a distributed environment between buyers and suppliers.
- **Community:** Building a sense of community is essential to attracting more buyers and suppliers to obtain critical mass. By providing the latest news, information and data, the marketplace will encourage regular visits. Also, real-time collaboration between trading partners significantly expedites the trading process.
- **Services:** Value-added service offerings will become the norm as digital marketplaces battle for business loyalty and a comparative advantage. The market maker has access to large amounts of product, transaction and market data that will be invaluable for trading partners to grow their business.

B2B Commerce Models Are Varied and Unique

There's no doubt that the ubiquity of the Internet has created what seem like endless opportunities and has resulted in some interesting

and creative business models. Thousands of marketplaces have evolved, each with its own approach, implementing multiple revenue models including auctions, reverse auctions, independent exchanges and hubs. Forrester Research estimates that 75% of these marketplaces will provide more than one transaction mechanism within two years.

Complex Products and Business Relationships

In the B2C e-commerce space, the product remains fairly static and price becomes the differentiating factor. However, in the B2B space the product and transaction variables are much more complex. Products have multiple dimensions and origins, and an entire list of quality specifications. Transactions have to incorporate many variables, including payment terms and conditions, order status and shipment confirmation. In addition, buyers may use a marketplace to work solely with a group of preferred suppliers and to trade on prenegotiated contract terms. The list continues to grow...


Interoperable Marketplaces

The rapid growth of the Internet has resulted in an uncoordinated approach to the adoption of Internet business models, resulting in complex integration and interconnectivity issues between trading partners. The number of marketplaces is expected to reach 100,000 in the next three to five years. The very trend that aims to centralize fragmented markets will be threatened by fragmentation itself. Take the example of ATM networks, which were forced to integrate with one another to serve the needs of customers. Marketplaces will be required to do the same, so market makers should plan accordingly. The eCo Framework from CommerceNet offers an XML-based solution that enables businesses to simply register their product and service offerings in order to participate in a market without requiring them to change the way they do business.

Open, Standards-Based Infrastructure Solutions

To effectively support these numerous product attributes, complex transactions, multiple e-commerce models, unique business relationships and marketplace interoperability, a digital marketplace must fundamentally be very flexible. Previous buy-and-sell-side solutions were too rigid, took a long time to realize a return on investment and couldn't easily adapt to an organization's business processes. Market makers will look to open, standards-based solutions to protect their investment and provide the flexibility to adapt the marketplace to meet future e-business needs. Java, XML and messaging-based middleware will be the enabling technologies to effectively exchange data and information in real-time between trading partners and their existing systems. In addition, market makers should look to the eCo Framework from CommerceNet to empower marketplaces to interoperate with one another.

• • •

Digital marketplaces are rapidly on the rise and here to stay. To learn more about them and how your organization can benefit either as a market maker or trading partner, follow the following links: www.netmarket-makers.com, <http://eco.commerce.net> and www.fiorano.com. 

AUTHOR BIO

Alex Roedling is director of marketing at Fiorano Software, an e-business infrastructure solutions company. He holds an MBA from the Monterey Institute of International Studies.

alexr@fiorano.com

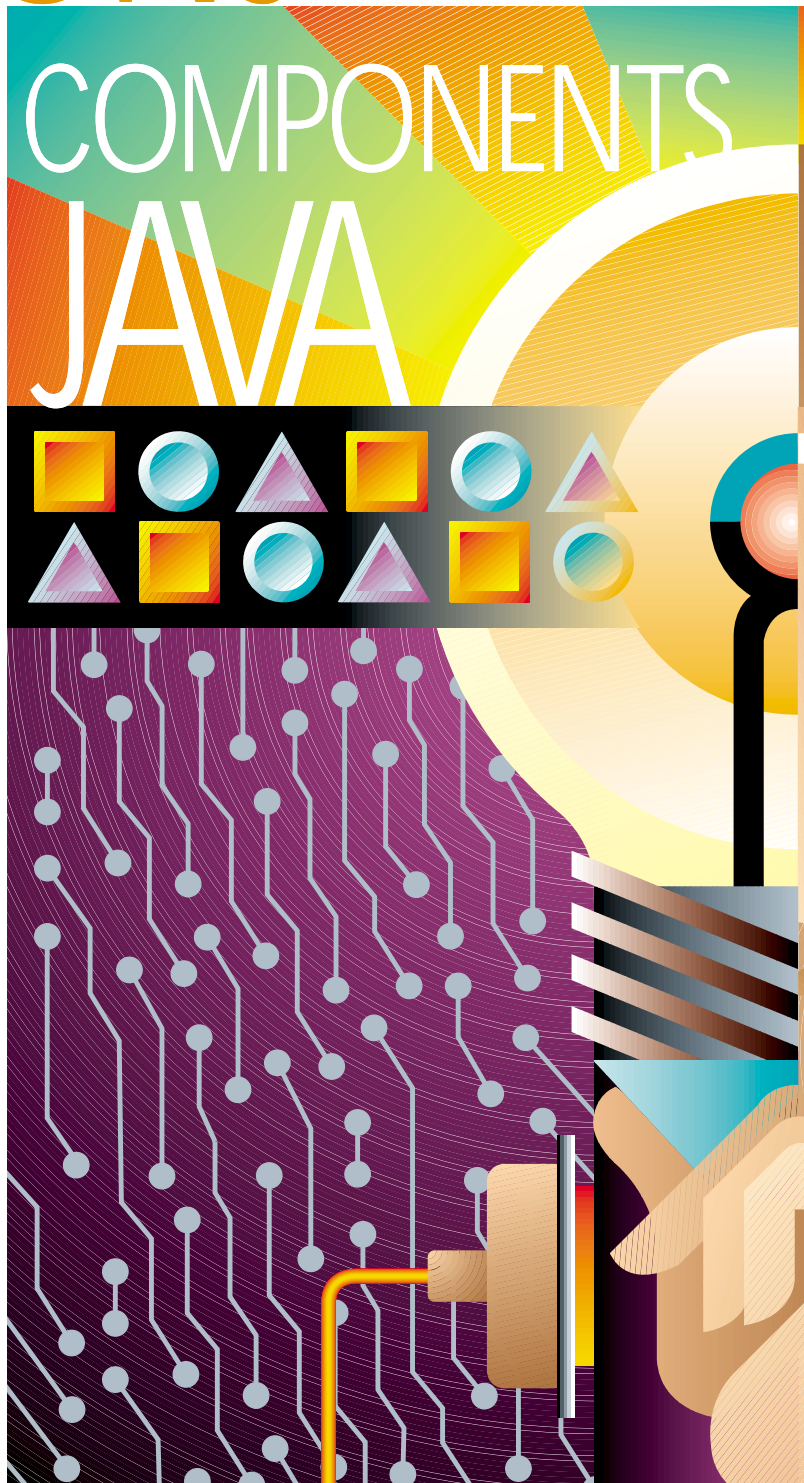
Insignia Solutions

www.insignia.com/jeode

BUILDING E-COMMERCE APPLICATIONS USING BUSINESS COMPONENTS FOR JAVA

WRITTEN BY JIM MILBERY

You can leverage the power of an extensible framework to construct a complete application with a JSP front end



The fifth anniversary of the inaugural JavaOne conference is upon us, and there can be little doubt that Java has had a profound impact on the way that applications are developed and deployed.

Over the past few years, Sun has continued to refine the Java specification, culminating in the groundbreaking release of the Java 2 Enterprise Edition (J2EE). Much of Sun's effort has gone toward the establishment of consistent application programming interfaces and technical infrastructures. The promise of multiplatform interoperability is certainly a heady goal, but it's a goal that seems more attainable now than at any time in recent memory. The combination of Enterprise JavaBeans, servlets and JavaServer Pages offers a powerful platform on which to build applications – but it may not be enough.



The IT industry is undergoing some powerful shifts that are beginning to garner press. The days of large, long-term projects lasting several years and with unlimited budgets – such as ERP implementations – are fast fading. Instead, corporations are favoring short (90-day) development efforts with near-term implementation goals. Large, complex applications are being built using a series of smaller projects that can be attacked in more realistic timeframes.

While this shift certainly improves the success-to-failure ratio of IT projects, it also requires developers to become much more disciplined about standards. After all, subsequent phases of a project must be able to build off the earlier phases in as seamless a fashion as possible. Corporations are finding themselves faced with the task of integrating their code into other applications managed by their partners and customers. This integration often involves the need to move applications to new platforms and environments. The result of these trends is a “Write once, build quickly and deploy everywhere” (WOBQDE) strategy that's quickly transforming the entire industry.

Many of the developers building these new applications will hail from the fourth-generation language (4GL) camp. While the J2EE product offers these developers a powerful platform for developing applications, it represents a fairly radical departure from the 4GL world. Java itself is a much lower level language for developing applications than traditional 4GL environments. Fourth-generation development environments (and languages) provide productivity improvements by handling many routine tasks such as optimistic locking, data caching and application deployment – freeing the developer to concentrate on business logic. The downside to most 4GLs has been their proprietary nature; developers needing to “go outside the bounds” of the language often find themselves faced with a formidable task.

Herein lies the problem. Is it possible to build enterprise-class applications in an incremental fashion along 4GL timelines using the J2EE? The answer just may lie with application frameworks.

Application frameworks are prebuilt libraries of core functionality used as a basis for developing higher-level software. Most object-oriented programmers tend to think of frameworks as nothing more than sets of class libraries but, while it's true they do generally include class libraries, they also provide tools to capture something of the flow of the business logic as well. While a class library that defines the attributes and methods of a “customer” object is certainly useful, a true framework will likely define how customers interact with other objects in the system.

The concept of frameworks isn't new, and many consulting firms and service organizations have developed their own custom frameworks in the past for other programming languages. Having prebuilt blocks of application logic available from project to project allowed these firms to reuse development work across multiple client companies. Instead of re-creating a solution for (say) order entry or error handling on a project-by-project basis, they built a framework for such tasks and used this framework in each individual application. With the popularity of the J2EE it's now possible to apply this same technique to the Java language.

Framework Definitions and Recommendations

There are no hard and fast rules as to what constitutes a framework. In general, it should provide the skeleton of an application and will either be horizontal or vertical in nature.

Horizontal frameworks provide a basic technical skeleton for operations such as:

- Locking (optimistic and pessimistic)
- Master/detail coordination
- Caching

Vertical frameworks on the other hand generally provide precooked business objects that may or may not include a technical skeleton. A vertical framework might include the following types of components:

- Inventory module
- Customer module
- Orders module

Informix/ Cloudscape

www.cloudscape.com

Informix/ Cloudscape

www.cloudscape.com

Horizontal frameworks provide the most flexibility. You can use the basic framework to build yourself a set of customized objects that can serve as your own "vertical" framework. Conversely, vertical frameworks are extremely effective for business problems that match the content of the framework. For example, if you need to build an inventory control system, you might find that a precooked inventory framework is a good way to jump-start your development effort. If you find yourself making extensive modifications to the base vertical framework, however, a horizontal model might be a better option – you can use a horizontal framework to build your own vertical framework.

In either case, you'll want to be able to modify the framework as necessary and you'll want to be able to upgrade the framework without making manual modifications to production applications that have been deployed using it. Let's say that the framework includes some base logic for caching data and that you use it to deliver your own order entry application. Over time the vendor improves the caching techniques in the framework. Ideally you'd want to be able to deploy the updated framework without having to modify your production application. At a bare minimum, you'd want to be able to have your specific modifications persist through an upgrade to the framework.

Historically speaking, this is one of the basic flaws in most 4GLs that can be addressed by a good Java-based framework. 4GLs provided productivity through their own proprietary framework managed by the language itself. While this generally made programmers more productive, there were almost always some severe limitations that couldn't be overcome. Since the framework was proprietary, there was often no way to modify or disengage it. With a Java-based framework it should be possible to overcome this problem. A good framework will allow you to override methods (or replace them) as necessary on a class-by-class basis.

It's important that any framework be tightly integrated with your development environment. While you'll want to have flexibility in deployment, it can be counterproductive for your programmers if they're required to "jump through hoops" to work with the framework. It's a double-edged sword: on the one hand, you'll want to be able to use the framework directly in your developer IDE because this will help you to maximize programmer productivity; on the other hand, you don't necessarily want to see all of the low-level details of the framework when you're debugging or testing the application.

Above all, you'll need the manner in which applications are deployed with the framework to be flexible. Frameworks too tightly bound to a single deployment methodology (or server) will cause headaches in the long term. In today's marketplace it's important to be able to address two critical server markets, EJB and CORBA. But while you may develop applications that are strictly tied to either EJB or CORBA, there's no guarantee that your business partners will follow the same model. Being able to deploy your applications into either model from a single framework can thus pay dividends down the road. You'll also need to access your framework objects from a variety of client applications including Java client applications, wireless devices, JavaServer Pages and Java servlets.

In the remainder of this article I'll be taking a more detailed look at frameworks using Oracle's Business Components for Java (BC4J) product.

Business Components for Java

Oracle's BC4J is a Java-based programming framework that enables programmers to develop, deploy and customize multitier database applications. It is intriguing in that it's actually a framework designed to create other frameworks.

A BC4J application comprises three tiers – client, business logic and database. The client tier can range from a full-blown desktop workstation running Java applications all the way to ultra-thin devices such as a PDA or cellular phone. The business tier contains business rules, application logic and predefined views of business data – and can be deployed as EJB session beans or CORBA server objects or used as a simple set of local classes. The database tier contains the persistent store for the application (such as an Oracle8i database).

BC4J is both a development framework and a deployment framework – all written in Java – and provides a suite of design-time wizards and editors providing you with the tools to define the base characteristics of your components. Each component (constructed of attributes, relationships and business rules) is then implemented as Java source code and XML metadata that implements the behavior you've specified during the design phase. The generated code inherits from the BC4J framework, keeping the Java source files relatively concise, thus making it easier to modify the components as necessary.

BC4J applications are constructed from six framework components (see Figure 1):

- *Application modules*
- *View objects*
- *Entity objects*
- *View links*
- *Associations*
- *Domains*

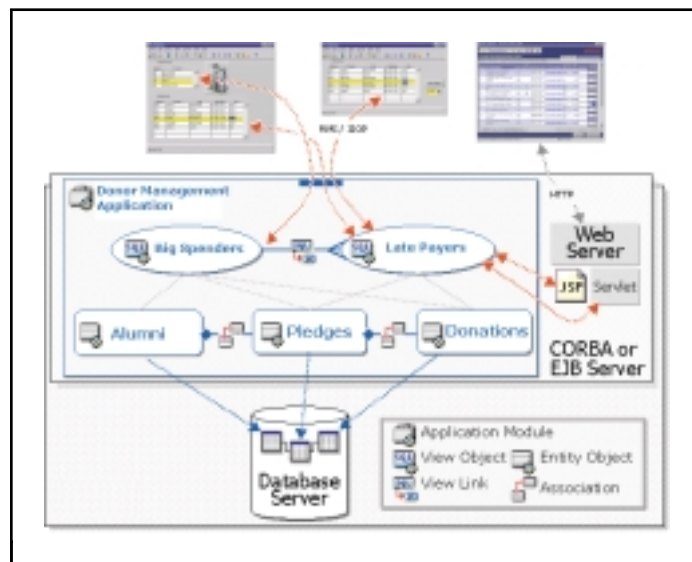


FIGURE 1 Sample BC4J application

Entity objects encapsulate the business logic and attribute (column) information for database objects. The underlying database objects themselves can be tables, views, synonyms or Oracle snapshots. Within the BC4J developer environment you can either define new entity objects and then use them to create new database tables or you reverse-engineer them from existing database objects. In the sample donor management application shown in Figure 1, *Alumni*, *Pledge* and *Donation* are all entity objects. By default the attribute names will match the table column tables, but you're free to change the attribute names to better reflect the needs of your business within the BC4J entity object. Since the entity objects are derived from database objects, they inherit database information such as primary/foreign key relationships, data type, length, precision and scale information.

Relationships between entity objects can be defined through associations. Consider the *Alumni* and *Pledge* entity objects in Figure 1. Although it's likely that a referential relationship already exists in the database between the *Alumni* and *Pledge* tables, you can define an association at the object level with BC4J as well. (In fact, BC4J will reverse-engineer these associations for you if they exist in the database.) Associations provide you with the tools to define accessor methods in the model. For example, you could create an accessor method in the *Alumni* entity object called *getPledges* that would retrieve the *Pledge* records automatically.

View objects are logical definitions of data that are used to join, filter, project and sort business data for a specified scenario. Although they're similar in concept to database views, they provide for a much higher level of abstraction. Figure 1 shows two view objects in the form of

KL Group Inc

www.klgroup.com/ticket

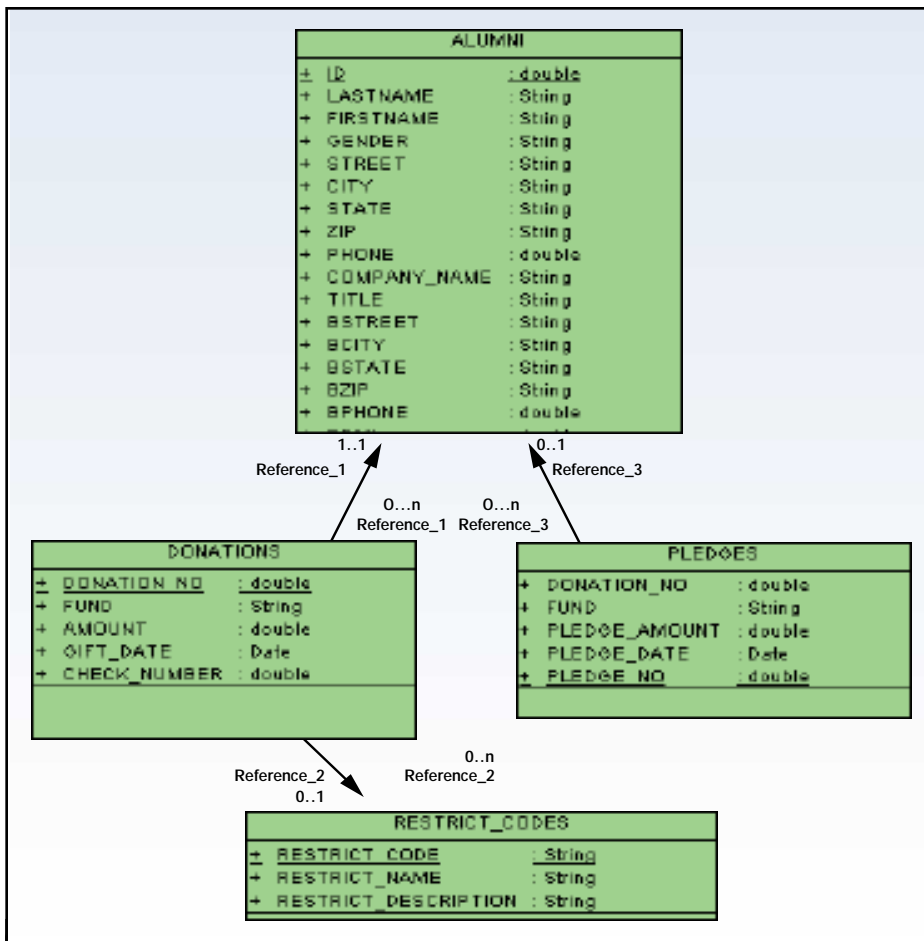


FIGURE 2 Alumni UML model

BigSpenders and LatePayers. View objects use SQL under the covers to fetch their data, but client applications are insulated from these details. Application code manipulates the data by “getting and setting” attribute values, and changes are persisted to the database by the framework when transactions are committed. When you define a view object you can determine which underlying attributes are read-only and which ones support read/write operations. BC4J caches view objects at the entity object level and all view objects referenced within a single transaction share the same cache, so that changes to one view are visible to all other views within the same transaction. Many business intelligence-style applications offer “click-through” capabilities for drilling down into detail data from master records and the view cache can simplify the development process for these applications.

BC4J allows you to define relationships between multiple views through the View Link component. The View Link wizard provides a tool for specifying the relationship, either one to one or one to many. View Links provide restrictions against detail data (much like adding a WHERE clause to SQL) and are particularly useful in designing master/detail and master/detail/master relationships in your application.

Data definitions and validation logic for attributes can be specified using Domains. The base Domain component is provided by the BC4J framework itself, but the developer defines the data type, Java class and validation built into the constructor. Many common validations (primary key, mandatory, unique, persistent and default value) are provided within the domain wizard. Domains can also serve as base data types for new attributes. For example, creating an “alumni ID number” domain in the donor management application.

The Application component is a Java class that inherits from oracle.jbo.server.ApplicationModuleImpl – and the Application wizard generates this class for you. The application component provides an overall container for your application. As with any framework, you’ll

View Objects support smart caching and pessimistic locking

All modifications (inserts, updates and deletes) to Entity Objects are handled as part of any transaction

The framework automatically manages SQL lookups as you traverse associations in code.

TABLE 1 BC4J Automated Functionality Examples

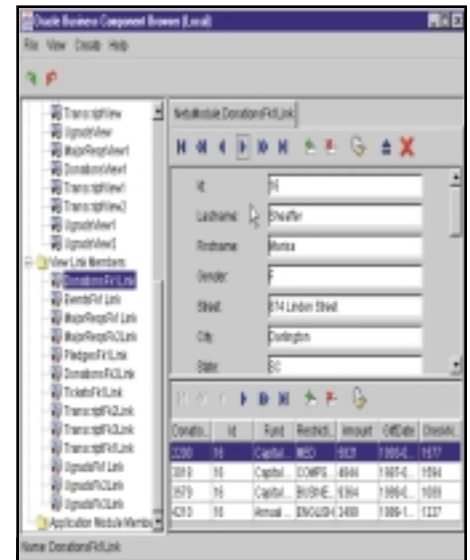


FIGURE 3 Alumni/Donations view

want to have the ability to customize this class. BC4J allows you to specify a custom class (which must extend from the ApplicationModuleImpl class) for this operation as well. If your organization already has some application-specific logic in place, you can use this technique to include your own logic in the generated framework. This is a critical area in which Java-based frameworks diverge from classic 4GL-style frameworks. Whereas a 4GL framework tended to force developers to accept the generated framework, the Java-based framework is extensible.

Working with BC4J

The BC4J is included with Oracle’s JDeveloper IDE. While you can create complete applications from scratch using a framework, many applications are constructed from preexisting data models. A horizontal framework can be particularly useful in such cases. Since you already have the data model in place, there’s no value in having pre-configured entity objects as you would with vertical frameworks. In-house we use an existing data model as a starting point for technical testing. This logical model is based around a fictional university database of student and alumni data. In order to get the most out of a framework it’s important to have a reasonably well-defined data model. For example, the NETU model (see Listing 1) makes extensive use of primary and foreign keys to map the relationships between the various tables. If we’d ignored declarative integrity constraints and built the data integrity into our application using database triggers and stored procedures, it would be much more difficult for the framework to interrogate the data model.

The NETU data model is composed of 15 tables, but the logical donor management application concerns itself with the Alumni section of the data model as shown in the UML diagram in Figure 2.

ThinWeb Technologies

www.thinweb.com

Each UML entity in the UML model corresponds to an entity object in the BC4J framework. From this simple data model you can see the power of using a development framework. The BC4J development environment created a complete application skeleton implemented in Java and XML from the data model. Many of the most common processing tasks have been implemented as a series of Java classes. For example, each Alumni record is connected to zero or more Pledge records that are in turn connected to zero or more Donation records. This is a fairly complex master to detail/detail relationship that would normally require a significant amount of basic coordination code to implement. This is exactly the situation in which a framework can provide significant value. In the case of BC4J, the framework includes a significant number of generated routines for working with your data such as those listed in Table 1.

One of the more interesting built-in capabilities is that multiple views of the same data within a single session are automatically synchronized. Figure 3 shows the test interface for the generated application with the Alumni table joined to the Pledge table.

The test interface provides a simple mechanism for working with the generated code in an interactive fashion. You can navigate through the Alumni records and they're automatically coordinated with the proper Donation records. Even if you open multiple Alumni views and modify data, it will remain consistent within all of the interfaces. Experienced 4GL developers are most likely accustomed to getting this type of default behavior out of their development tools, but this type of automated development might be new for Java developers. The key difference with the BC4J framework versus a proprietary 4GL is that the generated application is Java and XML. Changes to attributes are handled with get and set methods, and navigation through rows is handled by built-in navigation methods. The objects and attributes themselves are implemented as XML documents, as shown in Listing 1.

All of the attributes for each object are implemented as XML documents in the BC4J framework. Some of the elements in the XML descriptor will be immediately familiar, such as the base data type for each field in the table/view. BC4J implements all of the extended attributes in the XML file as well. Foreign key connections and field validations are also generated into the XML document for each component.

The combination of Java code and XML component definitions makes it easy to modify any behavior of the framework. Developers can also modify the application easily with minimal impact to the code by leveraging the Java language's "extend" capability. (You can also extend your business components by modifying the XML metadata.) As you roll out new versions of the base application, these customizations are preserved. Thus, if a client were to create a modified version of the Alumni table that supports additional attributes, new versions of the base application could be deployed subsequently without losing these customizations.

Deploying Applications

The last critical element of any framework is the ability to deploy the application in a flexible manner. While you might standardize on the J2EE as your deployment platform, you may need to deploy the same logical application as a thick Java client program using Java servlets and JavaServer Pages. You may also find it necessary to deploy the application to a CORBA environment in order to connect the application to an external application controlled by one of your business partners. For example, in the sample NETU case we might decide to use a third-party telephone calling service to contact the Alumni and solicit pledges and donations. While we may implement the donor management application as an EJB application internally, we could deploy portions of the application externally as a CORBA service (or as a set of local classes). Our telephone solicitation partner could connect to the Alumni data through their own internal CORBA applications (for example). Listing 2 shows the partial results of generating an interface for our NETU application as both an EJB application and a VisiBroker CORBA application.

All of the necessary code for the application is deployed along with the custom code. With BC4J the client code is designed to work against interfaces, so it's not necessary to modify developer-written application code when deploying the application to multiple tiers. The default deployment model with BC4J uses a very coarse-grained application model. In the NETU example case the application was deployed using a single remote interface for CORBA/EJB (see Listing 2). While you are certainly free to expose additional business methods as remote interfaces, all of the core functionality that we've discussed thus far is provided in the default BC4J framework.

Summary

In record time we were able to construct a complete application with a JSP front end against our NETU database model by leveraging the power of a framework. The application provided read, write, update and delete functionality across all 15 tables – including the coordination of multiple master/detail relationships, query by example and foreign-key validation. Using the BC4J framework provided us with the luxury of concentrating on the business logic without regard to the underlying plumbing. Unlike traditional productivity solutions, however, we can easily modify the base functionality of the application using standard Java coding techniques. Frameworks such as BC4J provide enterprise developers with the tools to tackle complex development efforts with short deadlines – a requirement of the new e-business economy. 🍌

AUTHOR BIO

Jim Milbery is a software consultant based in Easton, Pennsylvania, with Kuromaku Partners LLC. He has over 15 years of experience in application development and relational databases.

jmilbery@kuromaku.com

Listing 1: Donations Table in XML format

```
<?xml version="1.0" encoding='WINDOWS-1252'?>
<!DOCTYPE Entity SYSTEM "jbo_03_01.dtd">

<Entity
  Name="Donations"
  DBObjectType="table"
  DBObjectName="DONATIONS"
  AliasName="Donations"
  BindingStyle="Oracle"
  CodeGenFlag="4"
  RowClass="netu.DonationsImpl" >
  <DesignTime>
    <Attr Name="_isCodegen" Value="true" />
    <AttrArray Name="_publishEvents">
      </AttrArray>
  </DesignTime>
  <Attribute
    Name="DonationNo"
    Type="oracle.jbo.domain.Number"
    ColumnName="DONATION_NO"
    ColumnType="NUMBER"
```

```
SQLType="NUMERIC"
IsNotNull="true"
Precision="8"
Scale="0"
TableName="DONATIONS"
PrimaryKey="true" >
  <DesignTime>
    <Attr Name="_DisplaySize" Value="0" />
  </DesignTime>
</Attribute>
<Attribute
  Name="Id"
  Type="oracle.jbo.domain.Number"
  ColumnName="ID"
  ColumnType="NUMBER"
  SQLType="NUMERIC"
  IsNotNull="true"
  Precision="5"
  Scale="0"
  TableName="DONATIONS" >
  <DesignTime>
    <Attr Name="_DisplaySize" Value="0" />
  </DesignTime>
```


Allaire Corporation

www.allaire.com/download

```

</Attribute>
<Attribute
  Name="Fund"
  Type="java.lang.String"
  ColumnName="FUND"
  ColumnType="VARCHAR2"
  SQLType="VARCHAR"
  Precision="20"
  TableName="DONATIONS" >
  <DesignTime>
    <Attr Name="_DisplaySize" Value="20" />
  </DesignTime>
</Attribute>
<Attribute
  Name="Restriction"
  Type="java.lang.String"
  ColumnName="RESTRICTION"
  ColumnType="VARCHAR2"
  SQLType="VARCHAR"
  Precision="15"
  TableName="DONATIONS" >
  <DesignTime>
    <Attr Name="_DisplaySize" Value="15" />
  </DesignTime>
</Attribute>
<Attribute
  Name="Amount"
  Type="oracle.jbo.domain.Number"
  ColumnName="AMOUNT"
  ColumnType="NUMBER"
  SQLType="NUMERIC"
  Precision="10"
  Scale="0"
  TableName="DONATIONS" >
  <DesignTime>
    <Attr Name="_DisplaySize" Value="0" />
  </DesignTime>
</Attribute>
<Attribute
  Name="GiftDate"
  Type="oracle.jbo.domain.Date"
  ColumnName="GIFT_DATE"
  ColumnType="DATE"
  SQLType="DATE"
  TableName="DONATIONS" >
  <DesignTime>
    <Attr Name="_DisplaySize" Value="7" />
  </DesignTime>
</Attribute>
<Attribute
  Name="CheckNumber"
  Type="oracle.jbo.domain.Number"
  ColumnName="CHECK_NUMBER"
  ColumnType="NUMBER"
  SQLType="NUMERIC"
  Precision="4"
  Scale="0"
  TableName="DONATIONS" >
  <DesignTime>
    <Attr Name="_DisplaySize" Value="0" />
  </DesignTime>
</Attribute>
<AccessorAttribute
  Name="Alumni"
  Association="netu.DonationsFk1Assoc"
  AssociationEnd="netu.DonationsFk1Assoc.Alumni"
  AssociationOtherEnd="netu.DonationsFk1Assoc.Donations"
  Type="AlumniImpl" >
</AccessorAttribute>
<AccessorAttribute
  Name="RestrictCodes"
  Association="netu.DonationsFk2Assoc"
  AssociationEnd="netu.DonationsFk2Assoc.RestrictCodes"
  AssociationOtherEnd="netu.DonationsFk2Assoc.Donations"
  Type="RestrictCodesImpl" >
</AccessorAttribute>
<Key
  Name="DonationsPk" >
  <DesignTime>
    <Attr Name="_DBObjectName" Value="DONATIONS_PK" />
    <Attr Name="_isPrimary" Value="true" />
    <Attr Name="_isNotNull" Value="false" />
    <Attr Name="_isUnique" Value="false" />
    <Attr Name="_isCheck" Value="false" />
    <Attr Name="_isCascadeDelete" Value="false" />
  </DesignTime>
  <Attr Name="_isDeferrableConstraint" Value="true" />
  <Attr Name="_isValidateConstraint" Value="false" />
  <Attr Name="_isInitiallyDeferredConstraint" Value="true" />
  <Attr Array Name="_attributes" >
    <Item Value="netu.Donations.DonationNo" />
  </Attr Array>
</Key>
<Key
  Name="DonationsFk1" >
  <DesignTime>
    <Attr Name="_DBObjectName" Value="DONATIONS_FK1" />
    <Attr Name="_referencedKey" Value="ALUMNI_PK" />
    <Attr Name="_isPrimary" Value="false" />
    <Attr Name="_isNotNull" Value="false" />
    <Attr Name="_isUnique" Value="false" />
    <Attr Name="_isCheck" Value="false" />
    <Attr Name="_isCascadeDelete" Value="false" />
    <Attr Name="_isDeferrableConstraint" Value="true" />
    <Attr Name="_isValidateConstraint" Value="false" />
    <Attr Name="_isInitiallyDeferredConstraint" Value="true" />
    <Attr Name="_isDisabledConstraint" Value="false" />
  </DesignTime>
  <Attr Array Name="_attributes" >
    <Item Value="netu.Donations.Id" />
  </Attr Array>
</Key>
<Key
  Name="DonationsFk2" >
  <DesignTime>
    <Attr Name="_DBObjectName" Value="DONATIONS_FK2" />
    <Attr Name="_referencedKey" Value="RESTRICT_CODES_PK" />
    <Attr Name="_isPrimary" Value="false" />
    <Attr Name="_isNotNull" Value="false" />
    <Attr Name="_isUnique" Value="false" />
    <Attr Name="_isCheck" Value="false" />
    <Attr Name="_isCascadeDelete" Value="false" />
    <Attr Name="_isDeferrableConstraint" Value="true" />
    <Attr Name="_isValidateConstraint" Value="false" />
    <Attr Name="_isInitiallyDeferredConstraint" Value="true" />
    <Attr Name="_isDisabledConstraint" Value="false" />
  </DesignTime>
  <Attr Array Name="_attributes" >
    <Item Value="netu.Donations.Restriction" />
  </Attr Array>
</Key>
<Key
  Name="DonationsFund" >
  <DesignTime>
    <Attr Name="_DBObjectName" Value="DONATIONS_FUND" />
    <Attr Name="_checkCondition" Value="fund =
    &#39;Annual Fund&#39; or fund = &#39;Capital Cam-
    paign&#39;" />
    <Attr Name="_isPrimary" Value="false" />
    <Attr Name="_isNotNull" Value="false" />
    <Attr Name="_isUnique" Value="false" />
    <Attr Name="_isCheck" Value="true" />
    <Attr Name="_isCascadeDelete" Value="false" />
    <Attr Name="_isDeferrableConstraint" Value="true" />
    <Attr Name="_isValidateConstraint" Value="false" />
    <Attr Name="_isInitiallyDeferredConstraint" Value="true" />
    <Attr Name="_isDisabledConstraint" Value="false" />
  </DesignTime>
  <Attr Array Name="_attributes" >
    <Item Value="netu.Donations.Fund" />
  </Attr Array>
</Key>
<Key
  Name="SysC002937" >
  <DesignTime>
    <Attr Name="_DBObjectName" Value="SYS_C002937" />
    <Attr Name="_checkCondition" Value="&#34;ID&#34; IS
    NOT NULL" />
    <Attr Name="_isPrimary" Value="false" />
    <Attr Name="_isNotNull" Value="false" />
    <Attr Name="_isUnique" Value="false" />
    <Attr Name="_isCheck" Value="true" />
    <Attr Name="_isCascadeDelete" Value="false" />
    <Attr Name="_isDeferrableConstraint" Value="true" />
  </DesignTime>
  <Attr Name="_isDeferrableConstraint" Value="true" />
  <Attr Name="_isValidateConstraint" Value="false" />
  <Attr Name="_isInitiallyDeferredConstraint" Value="true" />
  <Attr Array Name="_attributes" >
    <Item Value="netu.Donations.DonationNo" />
  </Attr Array>
</Key>

```

Appeal Virtual Machines

www.jrockit.com

```

<Attr Name="_isValidateConstraint" Value="false" />
<Attr Name="_isInitiallyDeferredConstraint"
  Value="true" />
<Attr Name="_isDisabledConstraint" Value="false" />
<AttrArray Name="_attributes">
  <Item Value="netu.Donations.Id" />
</AttrArray>
</DesignTime>
</Key>
</Entity>

```

Listing 2: EJB Home and CORBA Stub for NETU

```

import netu.*;
import oracle.jbo.common.remote.*;
import oracle.jbo.common.remote.corba.*;

// --- File generated by Oracle Business Components for Java.

public class NetuModuleServerVB extends
oracle.jbo.server.remote.corba.vb.VBrokerApplicationModule {

    public NetuModuleServerVB() {
        init();
    }

    protected void init() {
        setRemoteInterfaceClass(RemoteApplicationModuleOpera-
            tions.class);
        setTieClass(_tie_RemoteApplicationModule.class);
        setApplicationModuleDefName("netu.NetuModule");
    }

    public static void main(String argv[]) {
        try {
            new
netu.server.vb.NetuModuleServerVB().initServer(argv);
        }
        catch (Exception ex) {

```

```

        ex.printStackTrace();
    }
}

package netu.common.ejb;

import oracle.jbo.common.remote.ejb.*;
import oracle.jbo.common.remote.*;

// --- File generated by Oracle Business Components for Java.

public interface NetuModuleHome extends javax.ejb.EJBHome {

    RemoteNetuModule create() throws java.rmi.RemoteException,
        javax.ejb.CreateException;

    RemoteNetuModule create(SessionInfo info) throws
        java.rmi.RemoteException, javax.ejb.CreateException;

    RemoteNetuModule create(RemoteApplicationModule parent,
        String amName) throws java.rmi.RemoteException,
        javax.ejb.CreateException;

    RemoteNetuModule create(RemoteApplicationModule parent,
        String amDefName) throws java.rmi.RemoteException,
        javax.ejb.CreateException;

}

```



GUEST EDITORIAL

Jeremy Allaire —continued from page 10

The Diminishing Role of Java

Ironically, as Java becomes an operating platform, more and more uses of the technology will diminish Java's visible role to the ultimate user. It's critical that developers and IT managers understand this shift and take off the Java blinders, as it were, to help them understand how to apply the platform overall.

Two key examples come to mind in which Java is the runtime platform but not the end-user technology: dynamic page engines and scripting environments, and XML protocols and Internet middleware.

DYNAMIC PAGE ENGINES AND SCRIPTING ENVIRONMENTS

Over the past several years a dominant model has emerged for delivering page-based Web applications. This page-based scripting model typically separates the task of dynamic content generation, basic user interactivity and simple application logic into a template- and script-based environment. Often this tier is combined with a business logic tier implemented using a component object model such as COM, CORBA or EJB, with the implementation environment typically C++ or Java.

Interestingly, the popular page-based scripting models (ColdFusion, ASP, JSP) are increasingly trying to abstract away the lower levels of complexity required by traditional system programming. In ColdFusion and JSP, for example, the ideal programming model becomes tags and tag libraries used by designers and interactive developers to put together a user experience. In this model, while Java may be the runtime platform (it is, in fact, the ideal runtime platform for this dynamic page tier), it's essentially invisible to the developer. And this is a very good thing. Java programmers should be happy to know that their platform is the foundation but that others are using it without the complexity of the full language.

XML PROTOCOLS AND INTERNET MIDDLEWARE

Another great example is emerging XML protocols for distributed application integration, including protocols for distributed objects, messaging, transactions and security. While J2EE provides an outstanding runtime environment for such protocols and interfaces, the use of these XML protocols will increasingly be divorced from developers knowing anything about the actual Java runtime. In most cases XML middleware will connect applications written in scripting environments such as JSP, ColdFusion, Perl and ASP rather than full-blown EJBs.

Both of these examples illustrate an ultimate paradox – which is that as Java becomes more and more successful as a platform, it will grow less and less visible as a language. It's critical that developers embrace this idea and open up to the fact that while not every application or system may be implemented in the Java language, ultimately it will still be running on Java bytecode.

The fact that these kinds of observations and discussions are happening symbolizes the incredible power and success of Java. With this power comes a degree of responsibility to vendors and customers with regard to redefining Java's role in computing. Whatever the case, happy fifth birthday! ☺

AUTHOR BIO

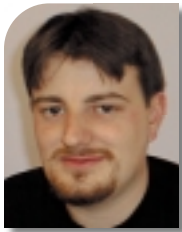
Jeremy Allaire is the cofounder and chief technology officer at Allaire Corporation, Inc. In this position Jeremy helps determine the company's future product direction and is responsible for establishing key strategic partnerships within the Internet industry. He is also the company's primary technology evangelist.

www.allaire.com

Compware NuMega

www.compuware.com/numega

Straight Talking. Two & JavaOne.Five



WRITTEN BY
ALAN WILLIAMSON

It seems to be the month for celebrations. Not only is it the fifth JavaOne, but it's also the second anniversary of this column. Both are still going strong, so hurrah! on both accounts. Long may it continue!

JavaOne is the Java event of the year. Period. It's our Oscars...our Olympics...our show. One of the few shows where the ratio of geeks per head is still encouragingly high. For those of you that haven't yet been to one, you're missing out on one great party. Now when I say *party*, I don't necessarily mean loud music and huge quantities of alcohol, but I mean party in the getting-to-know-everyone sense. I would of course be lying if I said that alcohol or music wasn't part of the equation!

We're currently preparing ourselves for the annual pilgrimage over the Atlantic to attend. Murray, The Riddler (Darren), Ceri and I will all be in attendance, hovering around the **JDJ** booth, conducting a demanding schedule of radio interviews. You'll recognize us fairly easily - we'll be the ones decked out in full Scottish kilts, with a microphone taped to our mouths. The **JDJ** staff and I are preparing the radio interview schedule, and let me tell you, we have some really cool interviews lined up, so be sure to stop by the booth and check us out. Sadly, my regular radio cohost won't be joining me this year; Keith has other duties to perform elsewhere.

Most of the **JDJ** editors will be there, so come and introduce yourself to us and tell us what you really think of our columns. But be warned: don't be too harsh - seeing grown men cry can sometimes be a hard thing to deal with. I'd especially like to extend an invitation to all those on the **Straight Talking** mailing list: I want to see faces. We're attempting to arrange a bit of a get-together, so be sure to stop by the **JDJ** booth and Web site for venue details.

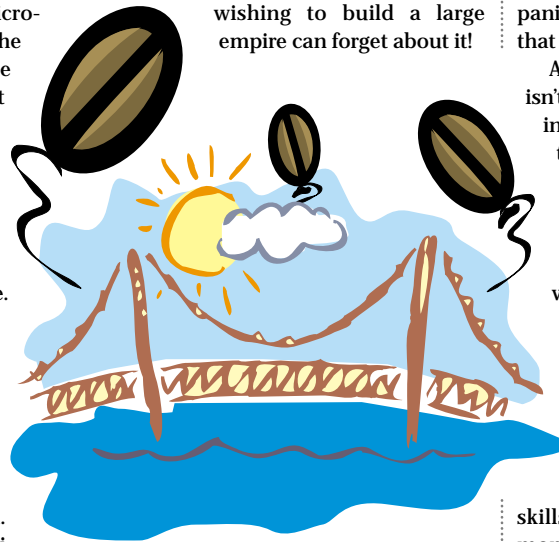
In the midst of all this excitement leading up to JavaOne, let me get on with business and move forward with this column. I have a number of things I want to bounce off you this month.

The "Valley"

Keith has just returned from a fact-finding mission to Silicon Valley. He went with a group of other Scottish

companies for a structured learning experience that was prepared by the Scottish Enterprise Board. Keith met with many representatives from all walks of the computing life, from venture capitalists right up to U.S. Customs officials regarding visas. He came back with many facts, some surprising, some not. Let me share some of them with you.

Funding...I'm not going to lie to you...it doesn't look good. It seems to be a very nepotistic world, and unless you know someone that can get your foot in the door, getting these people to even answer your calls is a major project in itself. If you do manage to get a contact, be sure to have your "exit strategy" figured out and ready to implement within at least two years. So anyone wishing to build a large empire can forget about it!



Assuming you've got some money sorted out, they'll want to put at least one person on the board and want you to have your offices no more than 30 minutes away from their offices. Hazarding a rough stab in the dark here, I'm thinking Scotland may just fall out with that particular criterion.

That said, there seem to be some shortcuts that can be employed successfully. If you're young enough that university is still in the cards, may I suggest you go to Stanford at Palo Alto? Since this has been the breeding ground for many of the large players in our industry, venture



capitalists are sniffing around this place like flies around dung. In one reported case a VC apparently snuck into a class that taught how to put successful business plans together. Upon class completion she canvassed some of the students for their business plans. Clever, clever.

Let's assume you've got your money, you've got your "exit strategy," and all you need to do now is to implement the idea. Oh boy, this is where the fun really begins. Keith spoke to many people on this front, including a number of companies looking to recruit and a number that offered recruitment management.

As regular readers know, this column isn't a great fan of the large salaries this industry is offering. The bubble on this particular minefield is going to burst and there's going to be such fallout from this over exaggerated worth. Keith was told one particular story of a 19-year-old who wanted well in excess of \$100K plus all benefits, stock options and so forth - and was then promptly shown the door. Now I'm sure some company will pay this foolish salary, but what are they really buying? Who knows? But one thing's for sure: there's a real skills shortage in the Valley at the moment, and with salaries rocketing forever upward, there seems to be no immediate end in sight.

Every cloud has a silver lining...even this recruitment one. Many companies are now looking outside the Valley for their development team. Many of the large start-ups have development departments dotted all over the globe: Norway, Scotland, India, Ukraine - to name but a few of the ones I know of personally. A budget of \$100K will buy you a team of world-class developers in any one of these countries, so why buy into the game of having to pay a single "boy" \$100K just because he's in the same location that you're in? It's madness.

Gemstone

www.gemstone.com/welcome

It's this very madness that has led to the boom in the software industry in India. I recently read an article on this topic that presented a well-balanced argument in favor of using such offshore development teams. Many are reluctant to entrust their ideas to teams that are so remote. That I can understand. But I'd strongly recommend that you look into some of these companies before you decide one way or the other.

Of course, I'm a big believer in offshore development; that's one of the services we provide here at n-ary. We've built the technology behind a number of the dot-com sites, all while being in the middle of the lowlands of Scotland and at a fraction of the cost that would have been incurred if bought locally. Testimonials from many happy customers state that by letting us handle the core technology, they can get on with making their business a success. Keith picked up many hot leads for us and we're evaluating them to decide which ones to explore further. That was another thing Keith noticed: the sheer volume of individuals with the latest winner. He has never seen so much hope in one place before.

For all those sitting with what you believe is a killer idea, best of luck to you. It appears to be a well-trodden path, with many others either in front of or just behind you. Much advice and at times conflicting advice is at hand. Keep cool and remain focused. It'll be worth it.

Bug Report

This column is written for your pleasure, and I welcome all suggestions and comments to keep it fresh and entertaining. With that in mind, a number of months ago I solicited the **Straight Talking** mailing list (http://listserv.n-ary.com/mailman/listinfo/straight_talking) for suggestions on what they'd like to see me addressing on a monthly basis. One suggestion that came back was to have a quick look at the official Java Bug Parade at the Developer Connection (<http://developer.java.sun.com>). I've been mulling over this idea and now, after polling a number of others, I hereby officially announce the inclusion of a new section, aptly titled "Bug Report." (Up all night agonizing over that title!)

I've spent quite a bit of time going through many of the bugs/RFEs (Request for Enhancements) that appear in the top 25 lists. Some interesting information for you: of the top 25 bugs, 20 of them are related to GUI APIs. I found this very surprising. Looking through the bugs didn't provide me with much entertainment, but flip over to the RFE and you find some really heated discussions.

At the top of the list, as expected, is the old JDK-on-UNIX chestnut. With over 2,900 votes, there is a call for an official port of the JDK to a FreeBSD platform. Apparently there's a workaround to allow the JDK to run on a BSD platform, but this hasn't appeased the masses. Considering the request was posted only in November, the response has been fairly quick to amass this amount of feedback. Conversely, a quick surf over to the recently closed bugs will reveal the closing of the "Support JDK on Linux," which secured over 4,500 votes to make it happen. Granted, this post has been open since December 1997 and was only closed in December '99, two years later. So all those on the BSD list...hang in there, it could be a long wait.

Moving on to an RFE that we here at n-ary would dearly love to see: the ability for Java to support the ICMP protocol. If you look at BugID#4093850, it contains many good discussions regarding why it should be included in the core JDK. For those of you that are unfamiliar with the ICMP protocol, it is the protocol that the likes of ping and traceroute use. It's part of the IP protocol and technically there's no reason why it shouldn't be supported. The good news on this front is that the necessary APIs that will allow access to this lower-level information could be provided in the next release of Java, code-named Merlin.

After reading the details on what could be included in Merlin, (http://java.sun.com/aboutjava/communityprocess/jsr/jsr_059_j2se.html), it would appear that another highly voted for bug could be knocked on the head. BugID#4075058, "Adding Support for Non-Blocking I/O," has gotten over 400 votes and has been on the list since August 1998. One major problem with handling client connections in Java is the requirement to have one thread per client connection to effectively handle communications. There are a number of workarounds, but none of them provide the required level of control. To this end, building applications that need to handle large volumes of clients – for example, Web servers – is limited to around 2,000 concurrent clients. This would appear to be the maximum number of sockets that Java can handle sitting in a read() blocking call with the bug report sighting a piece of sample code to prove this figure. What I found particularly amusing about this mail was just how passionate some of the comments at the bottom appeared. If you're bored one day, I'd recommend you read them – might prove interesting.

If you spend any length of time on this part of the Internet, it would be easy to become very cynical about the whole Java revolution. Reading through all these bug reports it's a wonder any of it

is working. Java is now evolving into a large beast, and with this a number of wee idiosyncrasies are to be expected. So don't panic just yet.

Support Hall of Shame

Here's an update on the Dell ASP fiasco I reported a couple of months back. You're going to love this, and although I have been sworn to secrecy concerning the originator of the post, it doesn't detract any from the story. You may recall my writing about how awful Dell's Web site was, that it resulted in our order being doubled and that it was due to their Web site crashing at the wrong time, thanks to the wonders of Microsoft's ASP. Well, this story far outranks my tale, and I implore you to be more creative in your custom choices when choosing your next system. Let me explain.

Reader-J (what we'll call him) wished to buy a multiple processor system with the Windows 2000 operating system (now do you see why I'm withholding his name?). He selected a 733MHz as his first processor, and his choice for the second processor was quite cool. In the drop-down he was able to select a second 733MHz at a reduction of \$349. Yup, a reduction. Cool, eh? The reason is that Windows 2000 apparently doesn't support multiple processors and the business logic at Dell couldn't cope with this and therefore offered a reduction. Reader-J had to haggle with the Dell salesperson before they'd honor the sale, but honor it they did.

We can't put this down to ASP, of course; it could just as well have happened with any other solution. It was the business logic that was screwed up, not the technology. As the complexity of configurations increases, so does the number of potential bugs. From a consumer's point of view, I'd recommend that you play around with some of these vendor Web sites. Who knows? You may end up with a configuration where they owe you money! Stranger things have happened.

Keep Those Stories Coming

On that note I'll bid you farewell for another month, and I'll look forward to meeting some of you at JavaOne.Five, so be sure to stop by the **JJJSYS-CON Radio** booth. ☺

AUTHOR BIO

Alan Williamson is CEO of n-ary (consulting) Ltd, the first pure Java company in the United Kingdom. The firm, which specializes solely in Java at the server side, has offices in Scotland, England and Australia. Alan is the author of two Java servlet books, and contributed to the Servlet API. He has a Web site at www.n-ary.com.

alan@sys-con.com

Cerebellum

www.cerebellumsoft.com

How to Control a Robot Over the Internet

Simple robotics with Java servlets

WRITTEN BY
DARREL RIEKHOF
& KEITH FUGG



So you want to build a robot that walks around and bumps into things. But that's not enough for you (this is the year 2000, after all); you also want to control your robot over the Internet. What's more, you want to use Java to control it.

We'll start by showing you the basics of controlling an electric motor over the Web. Specifically, we'll describe how to control radio control (RC) servo motors over the Web. Servos are electric motors popular for use in many applications, including robotics. The components of the system (see Figure 1 for an overview) include a front-end HTML form for entering commands to control the robot, and a Java servlet that accepts these commands and uses a free open-source Java software kit from FerretTronics to send them through a serial port to your robot. On the other end of the serial port receiving these commands will be a FerretTronics chip, the FT639 Servo Controller. The FT639 converts the serial data into electric pulses that a servo can understand. The FT639 sends the signal to the appropriate servo, which responds by moving to the position specified. We'll assume that your robot is attached to your serial port and that the computer you're using is running a Web server that supports Java servlets.

Software

The software for controlling the robot has four parts: an HTML form, a Java servlet, the FerretTronics Java Development Kit (FTJDK) and JavaComm. These four components are listed in the same order that commands flow through the system: you enter the commands in the HTML form, then press the send button and they're posted to a Java servlet that uses the FTJDK to format the commands and send them on through the serial port using the JavaComm package (`javax.comm.*`) – the standard Java extension for communication ports.

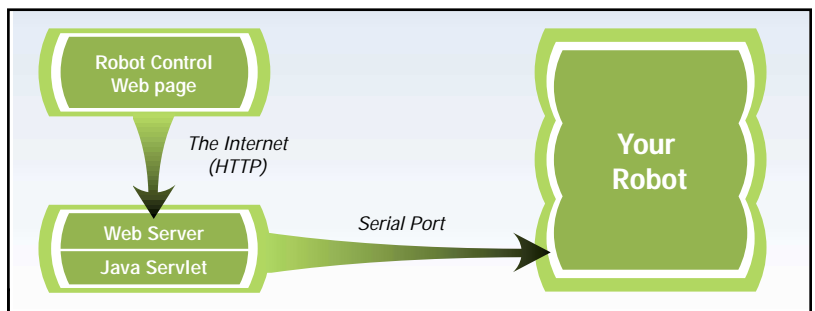


FIGURE 1 Overview of system

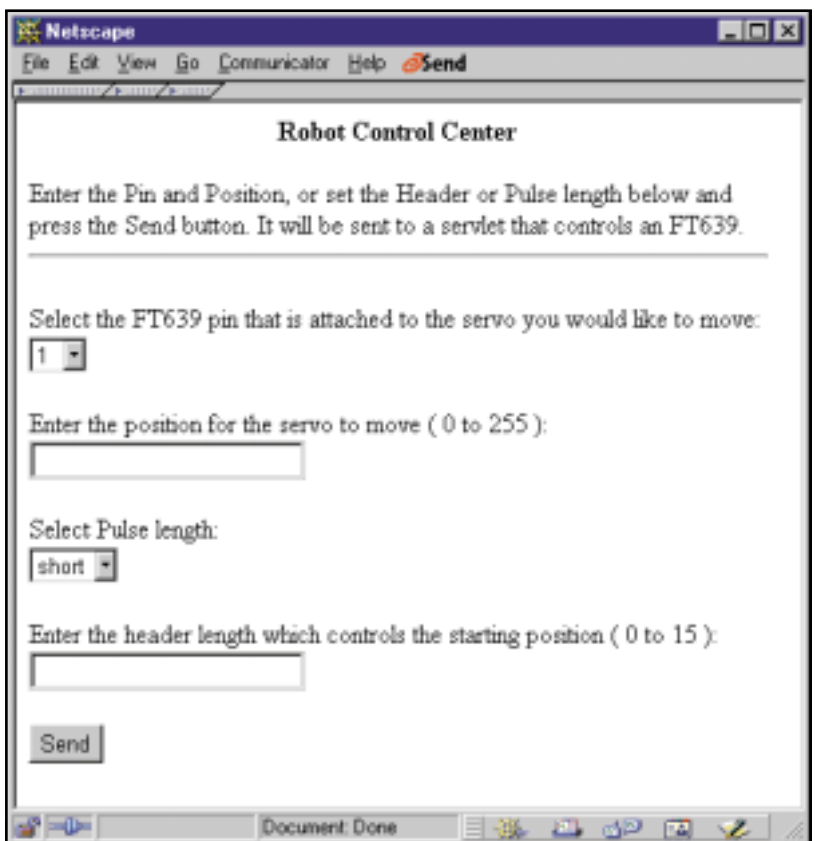


FIGURE 2 HTML form, the robot control center

VisiComp

www.visicom.com/jdj6

HTML Form

Before anything, we need a way for a remote user to enter and send control commands to the robot. We chose to use a Web browser and an HTML form to post the commands to a Web server (see Figure 2). We could have chosen to use an applet to make a niftier graphical user interface for controlling the robot, but opted for standard HTML to keep things simple.

The form has standard HTML controls for setting items, as follows:

- **Position:** Moves the servo to a certain position.
- **Pin:** Specifies which servo to move. An FT639 chip has five pins and a servo could be attached to any of them.
- **Pulse length:** Sets the number of degrees the servo can rotate. If pulse length is set to short, the servo can turn 90 degrees. If it's set to long, the servo can turn 180 degrees.
- **Header length:** Adjusts the zero position of the servo.

Servlet

After you press the send button on the HTML form, all the information is posted over the Internet to the Java servlet (see Listing 1). The servlet is installed on a Web server and the Web server is physically attached to your robot through a serial port. We're keeping this servlet simple, with limited error-checking, and avoiding performance issues. The servlet works, but isn't robust or efficient.

A few things will need to be set up on your Web server to get this servlet working. First, you'll need to get Java's serial port package (JavaComm). It's a standard Java extension package (javax.comm.*). The Windows and Solaris reference implementations are available on Sun's Web site (www.sun.com). Implementations for other operating systems are available, but you might have to dig around if you're using an operating system other than Windows or Solaris.

Three things you need to remember:

1. Put the comm.jar file from the JavaComm package in the servlet engine's classpath.
2. Ensure that the comm.jar file knows how to find the native code that's writing to the serial port (a DLL file on Windows).
3. Check that the JavaComm code has access to its properties file, javax.comm.properties, which comes with JavaComm.

If these steps aren't followed and JavaComm isn't set up correctly, it won't be able to find any ports on your system and the servlet won't work. (See the documentation that comes with JavaComm and your servlet engine for more details.)

The final thing you need to get is the FTJDK – it's available at www.ferrettronics.com. You'll need to put the ftjdk.jar file in the servlet engine's classpath.

If you're not using Windows or if your robot isn't attached to COM2, you'll need to adjust the servlet code accordingly so that it accesses the correct serial port on your system, and then recompile the servlet like this:

```
if ( portId.getName().equals( "COM2"
) ) { ... }
```

Let's take a look at the servlet code, starting with the doPost method because that's the method that gets invoked when a remote HTML client presses its send button and posts its form data to the servlet. The first part of the code deals with finding and opening the serial port. It iterates through all the communication ports it finds on the system and stops when it finds the one specified. You may need to change this code to point to the port your robot is attached to (COM1, /dev/term/a, and so on). Next, the servlet code gets the form data, puts it into convenient variables, prints their values back to the HTML client and then calls the sendTo639 method. This method uses the FTJDK, to write the appropriate bytes to the serial port to control the servo.



A typical RC servomotor

Let's take a closer look at the sendTo639 method. After it formats the parameters for the FTJDK, it then creates an Ft639 object from the FTJDK, connects it to a SerialPort object from JavaComm and finally sends the appropriate commands to configure the FT639 chip or to move a servo attached to one of the FT639's pins. An Ft639 object from the FTJDK knows how to translate these requests into bytestreams that a real FT639 chip can understand. The Ft639 object sends these bytes to the connected serial port when one of its methods is called. That's all there is to it. The bytes will flow out the serial port and eventually reach an FT639 chip. The FT639 translates the bytes to electric signals that a servo can understand and sends these signals out on one of its pins. The servo will then move to whatever position the remote HTML client requested.

The FTJDK

FTJDK is special software that can translate your commands into a form that a FT639 servo controller chip can understand. It also contains classes for some of the other chips that FerretTronics offers. These include the FT649 chip that serves as a serial router and lets you control up to five devices through a serial port. The FTJDK also contains software for a switch input chip – the FT629, which helps get feedback from your robot – and support for the FT609, a stepper motor logic chip.

FTJDK Design

When designing the FTJDK, we decided it would be best to allow the developer to access the object that's attached to the device being controlled without having to worry about where the chip is in the circuit hierarchy of FT639s, FT649 router chips and other chips. This is important since an FT639 could be several levels away from the serial port via multiple nested FT649+ chips. We originally designed the FTJDK so that developers were required to indicate the path the command would need to take through the circuit, but changed it to glue devices together with proxies. The proxies remember the correct path and let developers call methods directly on the FT639 or another control chip. (This is attached to the electric motor or other device that you're controlling.)

The downside of this arrangement is that the user of the FTJDK needs to recreate the physical hardware connections in the software so that the objects know where they should forward commands. For a large array of FT649s and FT639s this may be somewhat burdensome, but we feel the initial work is well worth the effort as it removes the need to include the routing information in the command every time you want to move a different servo. To redress the shortcomings of this design, a future version of the FTJDK will come with a GUI allowing you to create your circuit visually and then generate the code that glues the devices together.

Figure 3 is a diagram of the entire system. On the left is the software that we've described above. On the right is the hardware we'll describe now.

The Hardware

Many of you may cringe at the thought of dealing with hardware and electronics, but don't worry. It's easy to build the electronics in this project – it requires just a few components: an FT639 and circuit board, two resistors, a

Information Architects

www.ia.com

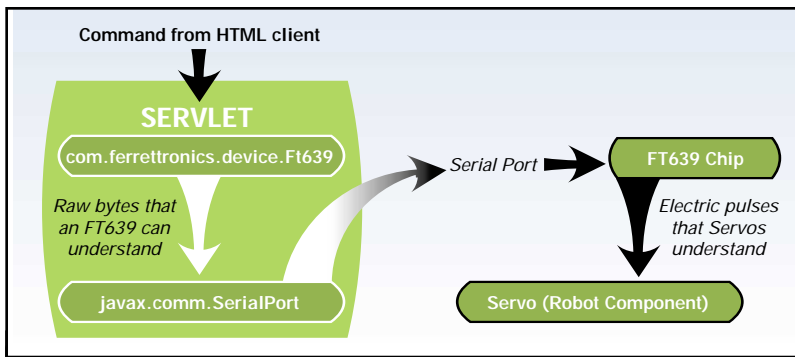


FIGURE 3 System diagram hardware

diode and a serial cable. We also sell a kit with step-by-step directions that contains everything you need to build the simple robot described in this article.

The robot in this article consists of a single RC servo and isn't very powerful or interesting, but it could become so if you treat it as a building block. Our example can be extended to create just about anything. The servo is controlled via the FerretTronics FT639, which is in turn controlled by the Java servlet described above. The FT639 has eight pins – one for the ground connection, one for the positive power supply (+5 volts) and one that is connected to the transmit line of the RS-232 connector from the host computer (the serial line). The remaining five pins are used to control up to five servos.

The connection to the serial port is the most complicated aspect of the hardware. It requires two resistors that act to reduce the voltage from the serial port. Additionally, a single diode is placed in line with the serial line to prevent a negative voltage from the host computer entering the circuit. You also need to connect the ground from the serial port to the ground used by the FT639. Finally, the RS-232 protocol on the host machine must be set to 2400 baud, 8 data bits, 1 stop bit and no parity. Don't worry about setting the RS-232 protocol if you're using the FTJDK – it takes care of setting the correct protocol before it sends any data over the serial port.

RC hobby servos have three wires terminating at a single connector. One wire is the ground, another is the +5 volt

supply and the third is for the control signal. The control signal line is connected directly to the FT639. The ground and V++ wires are connected to the same power supply as the FT639.

Conclusion

We've described a very simple – you might say useless! – application in which an RC servo is controlled via the Internet. The FT639 can control up to five RC servos, however, and the ability to control five servos adds a lot more flexibility to the design of a robot. Some examples of real-world projects you can make with this framework include a remote controllable Web-cam mount, a robotic arm, a six-legged robot, an XY plotter and any animatronic project like an animated skeleton, dinosaur or face. ☺

Resources

1. **FerretTronics:** www.ferrettronics.com
2. **Article Supplement:** http://weasel.ferrettronics.com/jdj/article_supp.html
3. **JavaComm:** www.javasoft.com/products/javacomm/index.html
4. **FTJDK:** www.ferrettronics.com/ftjdk.com

riekhof@primenet.com kfligg@azstarnet.com

Java Developer's Journal

www.javadevelopersjournal.com

AUTHOR BIOS

Darrel Riekhof, a Java consultant based in Tucson, Arizona, has worked for several companies including IBM, MCI, Blue Cross Blue Shield Association and Intel. He's also an owner of FerretTronics, Inc.

Keith Fligg, who has been interested in electronics and robotics since childhood, currently works as a system architect using the Shlaer-Mellor Method, building real-time embedded software architectures in C++ and Java. He's also part owner of FerretTronics, Inc.

Information Architects

www.ia.com

Listing 1

```

/**
 * @(#)Ft639Servlet.java
 */

import com.ferrettronics.device.Ft639;
import java.io.*;
import java.util.*;
import javax.comm.*;
import javax.servlet.*;
import javax.servlet.http.*;

/**
 * This is a sample servlet that takes input from
 * a form, parses and processes it, and sends
 * commands to a serial port to control
 * an FT639 and attached servos.
 */
public class Ft639Servlet extends HttpServlet
{
    /////////////// Data

    SerialPort serialPort = null;

    /////////////// Methods

    public void init( ServletConfig config )
    throws ServletException
    {
        super.init( config );
    }

    /**
     * Send data to port in response to the POSTed
     * form. Write a "confirmation" to the client.
     */
    public void doPost( HttpServletRequest req,
        HttpServletResponse res )
    throws ServletException, IOException
    {
        // Set the "content type" header of response.
        res.setContentType( "text/html" );
        // Get response's PrintWriter to return text
        // to the client.
        PrintWriter toClient = res.getWriter();
        // Find the serial port that the FT639 is
        // attached to. In this example, it is
        // attached to 'COM2'. We enumerate through
        // all the ports, and stop when we find it.
        Enumeration portList =
            CommPortIdentifier.getPortIdentifiers();
        CommPortIdentifier portId = null;
        while ( portList.hasMoreElements() )
        {
            portId =
                (CommPortIdentifier)portList.nextElement();
            if ( portId.getPortType() ==
                CommPortIdentifier.PORT_SERIAL )
            {
                // Windows
                if ( portId.getName().equals( "COM2" ) )
                {
                    try {
                        serialPort =
                            (SerialPort)portId.open(
                                "Sample639", // App Name
                                2000 ); // Timeout
                        break;
                    } catch ( PortInUseException piue ) {
                        toClient.println(
                            "Exception:<br> " + piue + "<p> " );
                    }
                }
            }
        }
    }
}

```

```

        piue.printStackTrace();
        System.exit( -1 );
    }
} // end if
} // end while
} // end while
// Get pin.
String strPin = null;
String [] pinArray =
    req.getParameterValues( "pin" );
if ( pinArray != null ) strPin = pinArray[0];
// Get pos.
String strPos = null;
String [] posArray =
    req.getParameterValues( "pos" );
if ( posArray != null ) strPos = posArray[0];

// Pulse Length and Header Length support
// not included in this code.

toClient.println( "<html>" );
toClient.println( "<title>Got it!</title>" );
toClient.println( "Pin=" + strPin + "<p>" );
toClient.println( "Pos=" + strPos + "<p>" );
toClient.println( "Serial Port = " +
    serialPort + "<p>" );

try {
    sendTo639( strPin, strPos );
} catch ( Exception excp ) {
    toClient.println(
        "Exception:<br>" + excp + "<p>" );
}

toClient.println( "</html>" );
// Close the writer; the response is done.
toClient.close();
if ( serialPort != null ) serialPort.close();
}

/**
 * Send data to 639.
 *
 * @param sPin Specifies the 639 pin that the
 * servo is attached to.
 * @param sPos Specifies the position to move
 * the servo to.
 */
public void sendTo639( String sPin,
    String sPos )
throws Exception
{
    if ( serialPort == null )
    {
        throw new Exception(
            "Serial port is null!!" );
    }
    int pin = Integer.parseInt( sPin );
    int pos = Integer.parseInt( sPos );

    // Some error checking on pin and pos values
    // could go here!!

    // Create a 639 and
    // connect it to the serial port.
    Ft639 ft639 = new Ft639();
    ft639.connectTo( serialPort );
    // Move the servo
    ft639.setServoPosition( pin, pos );
}

// EOF

```



Information Architects

www.ia.com

WRITTEN BY ERKEHARD ROHWEDDER

Embedded in Java PART 2

Mixing the Worlds of Java & SQL

Continuing our miniseries on SQL,
the standard for embedding
database SQL statements
in Java programs

It began sometime in late '96 or early '97 – JDK 1.0 still ruled and Tandem was still called Tandem, not Digital or Compaq – when people from IBM, Tandem and Oracle met and started to muse.

“Wouldn't it be nice to have SQL embedded in Java just as it's embedded in other host languages? However, we don't just want to copy previous efforts but to do justice to the Java language.” Of course, the embedding would have to permit the use of compiled SQL statements and be just as portable as Java code. It would also need to provide the easiest, most robust way to write SQL code in Java. Sun, Sybase and Informix soon joined the fray...and the “JSQL” effort was born.

“Gotcha,” you're thinking. “You meant to say *SQLI*, didn't you?” Well, yes and no – hold off just a second. The JSQL enterprise was all about *Java* programs that call *SQL*. When the same informal intercompany working group also embarked on an effort to describe the implementation of SQL stored procedures and functions in Java (christened *SQLI*, since it kind of goes in the opposite direction – *SQL* procedures that “call” *Java* in their body), this became known as *SQLJ part 1*. And when they started yet another project to describe how an SQL database could store Java objects in table columns and also publish them as SQL types, this was called *SQLJ part 2*. When the time came to submit JSQL to the ANSI standards committee, it turned out that the name was already a registered trademark of Caribou Lake Software for their JDBC drivers. So what did that bewildered bunch of computer scientists do when they realized they'd goofed? The same thing they've done since the dawn of the computer age: they started counting from zero! Thus *SQLJ part 0* was born.

On the other hand, the ANSI people don't refer to it as part 0; in fact, they don't even refer to it as SQLJ: to them it's *SQL Part 10: Object Language Bindings*. ANSI put its imprimatur on SQLJ part 0 around the end of 1998. Since then, SQLJ has been winding its merry way through the International Standards Organization, picking up a bunch of JDBC 2.0 features along the ride.

But enough history; let's get back to serious business. In this article I'm going to cover the following ground:

- A reprise of SQLJ iterators: all about positional iterators
- Calling stored procedures and functions in the database
- Reflections on bridging the gap between SQL types and Java types

As with Part 1 of this series of articles, I'll be giving various tips and exercises on the way through.

Information Architects

www.ia.com

MORE SQLJ SYNTAX

(See also Part 1 of this series [JDJ, Vol. 5, issue 5]).

- **Positional Iterators**

```
#sql modifiers iterator IteratorName(type1, type2, ...);
```

```
IteratorName iter; Initialize var1, var2, ...;
#sql iter = { SELECT ...}
while(true) {
#sql { FETCH :iter INTO :var1, :var2,...} ;
if (iter.endFetch()) break;
process var1, var2, ...
}
iter.close();
```

- **Stored Function Calls**

```
#sql result = { VALUES( stored_fun(:arg1, ...) ) };
```

- **Stored Procedure Calls**

```
#sql { CALL stored_proc(:invar, :OUT outvar, :INOUT inoutvar) };
```

- **PSM Set Statement**

```
#sql { SET :var = SQL Expression };
```

Get Into Position!

SQLJ provides two flavors of iterators. Last month we looked at named iterators, in which you specify both the Java column types and the column name. Remember that the name also appears as the accessor function with which you retrieve the column value. This kind of iterator is most “Java-ish,” and JDBC programmers immediately feel familiar with it.

Today I’ll be taking a closer look at positional iterators. They’re characterized by the order and by the Java types of their columns. Positional iterators require neither the next() method nor the accessors of the named iterator. They use a FETCH statement to advance to the next row and retrieve the column values into a list of variables all at once. Each variable in the INTO clause corresponds to exactly one column in the SELECT list in the same order. This will look familiar if you’re used to other languages with embedded SQL.

```
#sql { FETCH :p INTO :name, :salary };
```

Declarations for positional iterator types are even simpler than for the named variety.

```
#sql iterator PosIter (String, Double);
```

In the processing loop for a positional iterator, you issue FETCH statements to retrieve the next row of data into host variables. After a FETCH, the endFetch() call returns true if the FETCH was successful and false if there was no row left that could be fetched. Also remember to call close() on any iterator – named or positioned – once you’re done using it or you’ll find yourself running out of database resources. The following example uses a positional iterator:

```
String name = null;
Double salary = null;

PosIter p;
#sql p = { SELECT ename, sal FROM emp };

while (true) {
#sql { FETCH :p INTO :name, :salary };
if (p.endFetch()) break;
System.out.println(name + " would like to make " + (salary * 2));
}
p.close();
```

TIP: Even though it might look somewhat unusual, you should always employ the following template when using positional iterators:

```
Initialize var1, var2, ...
while (true) {
#sql { FETCH :p INTO :var1, :var2, ... };
if (p.endFetch()) break;
Process var1, var2, ...
}
```

Exercise: If you don’t follow the pattern above, things can go horribly awry. See what goes wrong with the following:

1. Move the endFetch() test after the processing of the fetched.
2. Use the following while loop condition: while (!p.endFetch()).
3. Don’t initialize the variables before the loop.

COMING SOON: SCROLLABLE ITERATORS

Scrollable iterators added in the ISO version of SQLJ are very similar to JDBC 2.0 scrollable result sets. To get one you just declare that an iterator implements the Scrollable interface:

```
#sql iterator ScrollIter implements sqlj.runtime.Scrollable
(String s, int i);
```

Given an instance siter of ScrollIter, you can use the familiar JDBC 2.0 movement commands:

```
siter.absolute(15); ... siter.relative(2); ... siter.last();
```

There’s also a positional, scrollable flavor, of course. It supports the corresponding FETCH syntax (FETCH ABSOLUTE, FETCH RELATIVE, FETCH LAST, ...), which does both movement and subsequent retrieval in one shot.

Let’s Get Results — Functions First

We’ve already seen how results can be received from an SQL statement when we used the SELECT-INTO statement. More often, results from an SQL operation are returned through an SQLJ assignment statement. Let’s look at a call to an SQL function SYSDATE():

```
java.sql.Date today;
#sql today = { VALUES( SYSDATE() ) };
System.out.println("The database thinks that today is "+today);
```

The VALUES(...) syntax is SQLJ-specific syntax for calling a stored function. Such functions might also take arguments, as in the following code snippet in which Next_Paycheck is an SQL stored function that returns the date of the next paycheck on or after a given date:

```
String moreMoney;
#sql moreMoney = { VALUES( Next_Paycheck(:today) ) };
```

(Note that we can receive an SQL DATE value in different formats in Java – in our examples, as a java.sql.Date and as a java.lang.String.)

Are We Outmoded Yet? — Getting Into Procedures

In the foregoing discussion we glossed over the fact that host variables or expressions are used in different modes:

- **IN:** The value of the expression is sent to the database.
- **OUT:** The expression denotes a location and receives a value from the database.
- **INOUT:** All of the above.

Host expressions, by default, have the mode IN – with the exception of host expressions in INTO-lists and the return value of a stored function call, which have the mode OUT. In all other cases you have to explicitly prefix the host expression with the mode. SQL stored procedures can have parameters with all three modes. The SQLJ syntax for calling a stored procedure is illustrated in the following code fragment:

InetSoft Technology

www.inetsoftcorp.com

```
int x = 10;
int y;
int z = 20;
#sql { CALL Toutes_Les_Modes( :x, :OUT y, :INOUT z ) };
```

TIP: You must add *OUT* or *INOUT* modes to all host expressions in procedure arguments that don't have the mode *IN*. Otherwise you won't see any values returned from the database in these positions. A good way to ensure that you've specified all the required modes is to run the SQLJ translator with online checking.

What Type Are You?

So far, we've used a bunch of Java types in our SQLJ programs without having a clue which types are permitted and how they're used. SQLJ includes all of the JDBC types with some additional twists. Following is a list of JDBC-supported Java types and how they're used in SQLJ. Please see the sidebar ("Coming Soon: Scrollable Iterators") for JDBC 2.0-specific type support.

- **Numeric types:** This includes: `int`, `Integer`, `long`, `Long`, `short`, `Short`, `byte`, `Byte`, `boolean`, `Boolean`, `double`, `Double`, `float`, `Float` and – just to prove I don't stutter – `java.math.BigDecimal`. So what's the deal with supporting both the primitive type (such as `int`, or `double`) and the corresponding Java object type (such as `Integer`, or `Double`)? In SQLJ the SQL `NULL` value always maps to Java `null` – and vice versa. Thus, if you retrieve an SQL `NULL` value into an `Integer`, you receive a Java `null`, but if you try to read it into an `int`, you'll only get an `sqlj.runtime.SQLNullException`, which is a subclass of `SQLException`.
- **Character types:** The Java type `String` represents these very well, thank you. Note that the Java `char` and `Character` types aren't supported by SQLJ or by JDBC (besides, they could only hold a single character, anyway). Also useful are the character streams `sqlj.runtime.ASCIIStream` and `sqlj.runtime.UnicodeStream`. One peculiarity about SQL is that columns defined as SQL `CHAR` type are automatically blank-padded. If you don't get the same string back that you inserted into the database, or if SQL comparisons with a given Java string fail mysteriously, you should check the SQL type used in your table.
- **Date and time types:** These include `java.sql.Time`, `java.sql.Timestamp` and `java.sql.Date`. Yes, that's `java.sql.Date` and not `java.util.Date` – don't confuse the two!
- **Raw types:** Raw data can be represented as `byte[]`, aka "byte-array," or – in stream form – as `sqlj.runtime.BinaryStream`, discussed next.
- **Stream types:** SQLJ provides new stream types `sqlj.runtime.BinaryStream`, `sqlj.runtime.ASCIIStream` and `sqlj.runtime.UnicodeStream` for wrapping a `LONG` (or `LONG RAW`) column in the database. These three stream types implement a `java.io.InputStream`. When you retrieve a `LONG` column, data in the same row prior to that column may be lost by some JDBC drivers (such as Oracle's). This imposes limitations on positional iterators (at most, one stream column is permitted, and it must also be the last column of the iterator) and it requires extra care when using named iterators (columns must be accessed in `SELECT` sequence). You could use byte arrays or `Strings` to circumvent these problems.

TIP: SQLJ (and SQL) perform quite a few implicit conversions between SQL and Java types. Although this can be useful, it may also lead to surprising and unexpected behavior. It is strongly recommended that you run the SQLJ translator online to check your program. However, the type checking this provides isn't adequate to guarantee the correct use of SQL types.

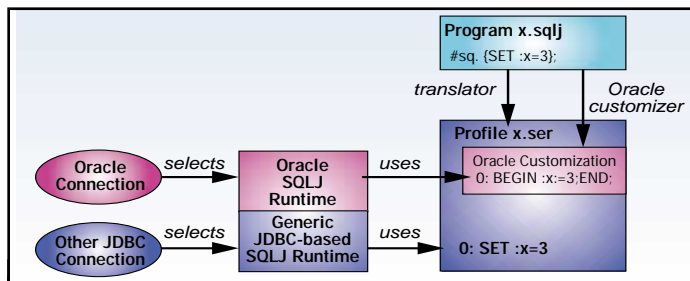


FIGURE 1 Profile customization

JDBC 2.0 TYPES IN SQLJ

The SQLJ ANSI standard is based on JDBC 1.2. The forthcoming ISO version of SQLJ, however, added support for new JDBC 2.0 types:

- **BLOB, CLOB:** Binary and character large objects
- **REF:** References to a `STRUCT` type
- **Named SQL types:** These SQL types use Java wrapper classes that know how to read instances of the type from the database or write them back. In JDBC the correspondence between wrapper classes and SQL type is kept in a `Map` object on the connection. In SQLJ the correspondence is provided through a list resource whose name is published on the connection context type. Three flavors of named types are supported:
 1. **STRUCT:** Structured SQL types. The Java wrapper class implements the `java.sql.SQLData` interface.
 2. **DISTINCT:** SQL distinct types that have an underlying build in SQL type. For example, a `SHOESIZE` based on a numeric SQL type.
 3. **JAVA_OBJECT:** SQL types based on Java types (see SQLJ, part 2). Instances of such types could, among other things, be stored as serialized Java objects.

Exercise: Investigate conversions between Java types and SQL types:

1. Take a positional iterator that contains a `String` column and an `int` column. What happens if you flip the corresponding host variables in the `FETCH` statement?
2. What happens if you flip the corresponding columns in the `SELECT` statement?

Get It Your Way – With Customization

Vendors need to be able to customize the way SQLJ programs are executed for their database. Take the following PSM set statement, for example.

```
SET :x = 2 + 2
```

Against an Oracle database, however, this would have to be written as follows.

```
BEGIN :OUT x := 2 + 2; END;
```

The Oracle customizer (see Figure 1) takes an existing serialized profile and adds Oracle-specific information to it – in this case the new SQL text. At runtime the actual database connection is used to determine which vendor's customization/runtime pair becomes activated. If no customization exists for a given connection, SQLJ reverts to using the standard JDBC API.

Okay, let's call it a day. Next time we'll cover (almost) everything else there is to know about SQLJ. I'll teach you some neat translator tricks for the command line. You'll also be initiated into the mysteries of execution contexts and connection contexts. Finally, we'll examine how JDBC and SQLJ can live in blissful harmony happily ever after. In the meantime, keep your feedback, your answers to exercises and your questions coming! 🍌

AUTHOR BIO

Ekkehard Rohwedder hacked on `ISQL` before he had to rename it to `SQLJ`. In his past he earned a `Dipl Inf` degree from `Friedrich-Alexander Universität Erlangen-Nuremberg`. Ekkehard leads `SQLJ` development at Oracle.

erohwedd@us.oracle.com

.SER FILE SECRETS REVEALED

Ever wonder what the `.ser` files contain? The serialized profiles provide full information on all of the (static) SQL statements. You can take a peek at what is in them with the `-P-print` command-line option.

```
sqlj -P-print *.ser
```

This lists for each SQLJ statement in your program:

- The SQL text of the statement
- The role of the statement and how it will be executed
- Descriptions of the parameters and of the result of the statement (if any), including their SQL and Java types

Cape Clear

www.capeclear.com

The Commerce in Java Application Servers

New developments in the marriage of Java application servers and commerce servers

WRITTEN BY
ANIL SAGAR



Before we start on the technical front, let me tell you about my latest acquisition. I recently went out to purchase a watch. My wife wanted me to buy one of the fancy ones, but I'm more excited by watches that have all the features – stopwatch, backlight, barometer, altimeter, everymeter – and the time displayed in BOLD DIGITAL NUMBERS. I usually end up using maybe three out of a hundred features, but at least I have them.

But I digress. I thought about where I could get the watch. The amazing thing is that you can buy them in nearly every kind of store – superstore, sports store, general merchandise stores, even gas stations. This is just an example of how the market grabs a commodity and merchants make it a part of their standard offering.

The Java application server market is no different. Application servers enable companies to build e-businesses by offering them the tools to do so. They started out by abstracting the interaction with the operating system from the application. Thus they created a new execution environment for industrial-strength distributed applications. The application developers were saved the pain of dealing with low-level system programming details and could concentrate on solving the business problem

This is because Java provides a virtual platform that is portable across different hardware platforms. Thus Java was used as a base for the object model (EJB) in the application server market. With its penetration into the middleware market, Java has also become the base for defining the APIs for the other app server services mentioned earlier. In fact, the term *Web application server* is almost synonymous with the term *Java application server* (Microsoft's application server suite notwithstanding).

The past year has seen a consolidation of several companies that offered different niche services in the app server market. You may have followed this thread of discussion in one of the previous *E-Java* columns. The current generation of Java application servers offers most of the following environments:

- Integrated development environments (IDEs)
- Scripting support
- J2EE development and execution environments

The Need for Commerce Servers

Existing application server environments allow application developers to develop the business logic for their specific applications. The next stage in the evolution of application servers involves the enhancement of the presentation tier of distributed applications. Scripting using JSPs, ASPs and other similar technologies allows the data generated in the business logic tier to be presented to the users of the application in a format specific to the business the application was built for in the first place. However, scripting is still too low level to completely design the front-end presentation tier. As the demands of the application become more complex, maintaining the content via scripting techniques becomes less manageable. These

demands arise as business processes are exposed to the user of the application, and an environment is needed for the user to have a richer interaction with the system. In addition, new requirements arise for the development of reusable components that can be applied to different businesses.

These requirements gave birth to commerce servers, a new breed of application server that offers services for implementing business processes. These servers are application packages that encapsulate functionality for commerce interaction such as buying and selling, collaborations, marketing, shopping and ordering, order management, and order tracking and fulfillment. Commerce servers combined with knowledge servers form the basis for creating portals and exchanges for dissemination of information. These server environments provide the means to enable personalization, membership registration, role-based access, usage trend analysis and market analytics. IBM's Net.Commerce, BroadVision's Enterprise server, Vignette's eBusiness Platform, Allaire Spectra, Microsoft Commerce Server and Oracle iStore are examples of commerce servers. Commerce servers help in developing customer-facing applications and are often found in front-office business technologies. Typical markets targeted by these servers are customer relationship management (CRM) and sales force automation (SFA) in the C2B space.

Commerce servers have existed in the application server market for a while, as stand-alone environments that can be combined with other Java application servers to provide complete customer-facing solutions. The next logical step in the evolution of Java application servers is for the application servers to offer a tighter integration with commerce servers. Going with the current market trend of acquisitions and merg-

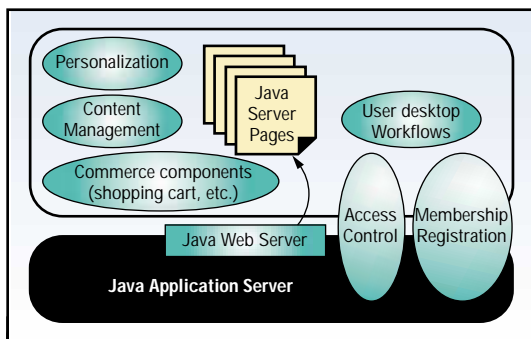


FIGURE 1 Commerce components using JSPs

on hand. As the application server market matured, several of these services became commodity services. The application server vendors embraced standards for the object programming models, transactions, Web access, security, and so on. Consequently, applications developed using application server services became portable across application servers. Java is responsible in a large part for enabling this "cross-appserver" application execution environment.

Hit Software

www.hit.com

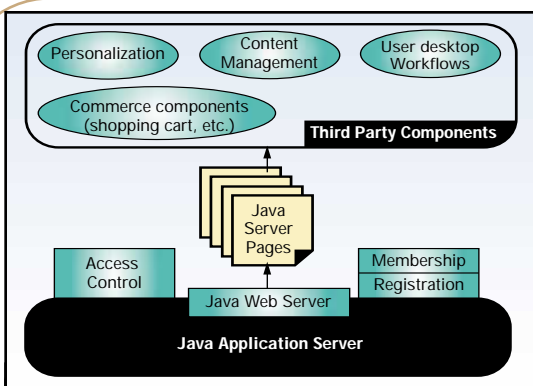


FIGURE 2 Third-party commerce components

ers, this means that the same vendor should offer traditional services offered by Java application servers as well as those offered by commerce servers. This evolution is already taking place. Several of the Java application server vendors are now offering commerce services.

Commerce Services

Before we go into specific examples, let's examine some of the common services usually associated with commerce servers. Typical Java application servers usually don't include inherent support for these services, though they often work in collaboration with products that do. Some services that should be provided by these servers are discussed below.

- **Directory Services:** Application servers often provide support for directory services, which are used for authentication and access control for the applications hosted by these servers. Most application servers that support directory services can work with an LDAP-based directory server, which is used for authenticating users and configuring groups with roles around partners and customers across multiple applications.
- **Role-Based Access:** Role-based security describes a set of services that allow you, the business manager, to create a set of access rules or set permissions to assign users to activities. Permissions enable different users (or roles) different access to content and activities in the system.
- **Membership Registration:** Membership registration is closely tied with directory services and role-based access. Membership registration is a feature of commerce servers that enables the creation of member groups for the purposes of authenticating and authorizing users for using the application in different ways. The registration service causes the members to get associated with a user

group, which in turn defines their access permissions and membership privileges. Thus the users get associated with a specific role.

- **Personalization:** Personalization is a concept that addresses the ability of an application to offer services customized to the user's preferences and business processes. Personalization includes the ability to perform dynamic profiling and target information to end users based on their interests and behavior. It's also used to track and store user information for use in the delivery of customized content and product information. Most personalization services offer rules-based personalization that implements publishing rules that deliver customized content based on user preference information expressed in the form of personalization rules.

Commerce Server Environments

Commerce servers offer programming and configuration environments used to define the business associated with the application. Some examples of such environments follow.

- **Content management environments:** Content management encompasses designing the content, logic, presentation and delivery of an application's Web front-end. This includes content syndication, which allows a business site to share content with other Web sites regardless of format or application environment. Outside content can be brought into the application. At the same time, a business's site assets can be exposed to site affiliates, allowing them to subscribe and define the delivery schedule and format.
- **Commerce component building environments:** Commerce servers offer sophisticated tools for building business components that interact with the underlying Java application servers. Such components include standard business components like shopping carts and electronic payment modules.
- **Messaging and workflow environments:** Workflow engines and messaging systems interact closely with application servers to provide an execution environment for a business process. Application servers provide an environment for creating abstractions to a particular business task inside a business process. Workflow engines provide a mechanism that allows nonprogrammers to define and model their business process. A

single business process leverages one or many business tasks encapsulated by an application server. Application servers use messaging systems to provide distributed asynchronous communications in a publish/subscribe environment, message queuing and guaranteed delivery of messages.

Java: Not a Panacea

Java enthusiasts may say, "But we can do all this in Java." This is certainly true. With the advent of J2EE, several of the business environments can be built using Java APIs and technologies. In fact, JavaServer Pages are touted by Sun as the solution to the presentation tier for e-business applications. The idea is that JSPs can be leveraged to create commerce components for customer-facing businesses. Figure 1 illustrates how Java application servers can use JSPs to create commerce components. JSPs will supposedly separate the programmers from the Web and business development teams. While true, the level of effort involved in creating such components and templates from scratch is substantial. In addition, the assumption is that, going with the open standards trend, these components will be reusable across different platforms running Java. The underlying assumption is that vendors will agree on what is a reusable component beyond the minimum standards and guidelines provided by Java. Although Java is certainly gaining the lead in the middle tier, it isn't the only player. Obviously, Microsoft and other technologies have a fair share of the marketplace. And while Java technologies suggest mechanisms for bridging with other technologies, most of the time this isn't a feasible solution in an enterprise-level solution.

The market already offers several choices in mature products that offer commerce components and services. A better choice for application developers is to combine these products with Java application servers to provide a comprehensive solution to their business needs. This is already taking place in the market today. Several businesses are building the middle tier of their applications based on Java technologies and are using third-party market leaders for other services needed by their applications. For example, Vignette's eBusiness Platform and BroadVision's Enterprise may be used to provide the commerce services that supplement the middle-tier services implemented in Java application servers such as BEA's WebLogic application server. Figure 2 illustrates how Java application servers can leverage third-party components.

Sybase

www.sybase.com/products/easerver

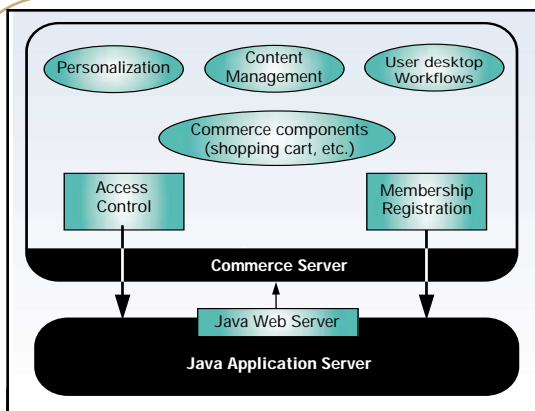


FIGURE 3 Java application server with commerce server

Java App Servers with Commerce Services

As mentioned earlier, the next logical step for Java application server vendors is to partner with commerce servers to offer comprehensive solutions for e-businesses. Going one step further, these vendors can combine to become one-stop shops for creating end-to-end distributed business applications. Figure 3 illustrates how a Java application server may be used with a commerce server. Such mergers and acquisitions have been in fashion for the past year and the combined product suites are already available in the market today. Some examples of these product suites are discussed below.

IPLANET ECXPERT, PORTAL SERVER AND IPLANET APPLICATION SERVER

The Sun-AOL (Netscape) alliance has been combining their product offerings to offer the iPlanet suite that offers products ranging from Web servers to e-business portals. Two of the recent offerings in this product suite are the ECXpert and the Portal Server. iPlanet ECXpert is an Internet commerce exchange application that enables an enterprise to automate and manage the processes that occur between organizations over the Internet and existing private networks. The iPlanet Portal Server is a portal platform that enables delivery of content, services, business processes and applications in personalized portals. Features offered by the Portal Server include community creation and management; multitiered portal personalization; creation of integrated content, applications and services through customizable portal channels; secure extranet access to portals by mobile/remote employees and suppliers; and integration with subscription-based external content from AOL/Netscape. These products leverage the iPlanet Application Server (formerly the Netscape Application Server) to provide a comprehensive suite of products.

AUTHOR BIO

Ajit Sagar is a member of a leading e-commerce firm in Dallas, Texas, focusing on Web-based e-commerce applications and architectures. A Sun-certified Java programmer, Ajit is also the editor-in-chief of SYS-CON's XML-Journal.

BEA WEBLOGIC COMMERCE SERVER, APPLICATION SERVER

BEA's traditional Java application server offerings have been enhanced with their commerce server, which enables deployment of personalized e-commerce applications that can be modified to meet customer demands or take advantage of new market opportunities. It provides online catalog, shopping cart, inventory management, order entry, order management and shipping components as well as a product recommendation engine that learns customers' behaviors over time. It offers business controls for marketing professionals that enable modification of the behavior of e-commerce applications that define interactions with individual customers – including what promotions they receive, what access they have and what content they see. The same controls allow product managers to manage product catalogs dynamically and change pricing policies at multiple levels. BEA's commerce server adds to their popular WebLogic Enterprise Java application server.

ALLAIRE SPECTRA, COLDFUSION AND EGIPT APPLICATION SERVERS

Unlike other vendors that started with base middle-tier functionality and worked their way to the front end, Allaire's products have expanded from Web authoring tools to a complete enterprise application product suite. Their flagship product is a non-Java application server (ColdFusion) that enables rapid deployment of Web sites. However, in the past year (and this one) Allaire has acquired technologies that should get them entry into the Java application server space, specifically LiveSoftware (JRun) and Valto (Egipt). Since Allaire started from the Web front-end, it isn't surprising that their commerce server, Spectra, is built in-house. Spectra is a packaged system for developing large-scale Web applications. Powered by ColdFusion application server, it spans three crucial application areas – content management, e-commerce and customer management. Spectra can be used to design the content, logic, presentation and delivery of the application, workflow and process flow automation, role-based security, personalization, business intelligence and data plus application syndication. Spectra uses the ColdFusion application server to get to its underlying Egipt Java application server.

Bluestone Total-e-Business Suite and Sapphire Java Application Server

Bluestone recently announced Total-e-Business product suite, a comprehensive e-business solution for deploying e-

business that addresses foundational e-business platform requirements including infrastructure, integration, content management and personalization. In addition, it delivers a set of e-commerce components typically required for selling goods and services on the Web – for example, catalog management, shopping cart and credit card processing.

SILVERSTREAM'S ADVANCED PORTAL FRAMEWORK AND SILVERSTREAM JAVA APPLICATION SERVER

SilverStream recently announced an advanced portal framework solution. Based on their SilverStream application server, it provides core features including Personalization, Content Management, a Component Framework and a library of commerce components. SilverStream's Portal Framework is a part of the SilverStream eBusiness Platform. The Portal Framework offers core commerce features such as user profiling, rules-based personalization, workflow management, caching and content management.

Trading Places

Back to my wristwatch. I ended up buying one with lots of cool features. The good thing about the wristwatch is that all the parts are developed in-house by the same vendor and are a part of the same product. Thus, if my "dual-alarm" feature breaks, I don't have to figure out how it interfaces with the rest of the cool features. The manufacturer knows how these things work together. However, in the case of one-stop shop application server vendors, it's a different story.

The hooks into the Java middle tier for all these commerce products are still the Java enterprise APIs. However, when you go out into the market for a comprehensive product suite, you need to be aware of some caveats. Most of these product suites have been packaged recently and are made up of several disjointed products. Just because a vendor offers the complete solution doesn't mean that the integration between the products is complete or painless. Vendors have gone through several acquisitions to complete the market story, to stay competitive and to drive up the price of their stock. Integrating these products into a single product offering will take time, effort and money. When you shop in the market for a product suite suitable for your needs, use due diligence to make sure you have integration support from the vendor. Sometimes it may be easier to buy mature products from separate vendors and do the integration as a part of your application development. ☺

ajit@sys-con.com

WebGain

www.webgain.com

A Java File Search Utility

It's functional as it stands — and you can add to it if you wish



WRITTEN BY
PAT PATERNOSTRO

File searches are traditionally accomplished by an operating system utility. Most operating systems provide some sort of search facility that allows the user to track down misplaced or forgotten files. However, the facilities differ in their approach for searching files – graphical versus command-line interface, comprehensive versus limited search capability.

If you work in multiple environments as I do, you become reliant on search functionality available in one environment that may not be supported in another. This article details a Java file search utility that provides a consistent user interface and consistent functionality when searching for files.

Interface

The FindFile utility is made up of two classes: FindFile and FindFileFrame, located in FindFile.java. The FindFile class (see Listing 1 on [JDJ](#) Web site) simply contains a main() method, which is the entry point for the application. The main() method instantiates the FindFileFrame class (see Listing 2 on Web site) where the utility's GUI is constructed and where the main processing logic is contained. The GUI is constructed using panels combined with three layout managers (java.awt.FlowLayout, java.awt.GridLayout and java.awt.BorderLayout) in favor of the more flexible, yet arguably more complicated and tedious java.awt.GridBagLayout layout manager. I work predominantly on Win32 operating systems (iWin9x, WinNT); therefore I modeled the FindFile utility's user interface (see Figure 1) after the Win32 Find utility (see Figure 2).

Implementation

The FindFileFrame class implements the java.lang.Runnable interface and two event listener interfaces: java.awt.event.TextListener and java.awt.event.ActionListener. The java.awt.event.TextListener interface method, textValueChanged(), enables and disables the Find button based on the length of text entered in the File Name field. This prevents the user from starting a file search without first typing in something for a filename. The Look In

field is optional and directs the utility to start the file search in the specified directory. If the field is left blank, the search is started in the current directory. The search is limited to the specified directory unless the Include Subfolders checkbox is selected. The user is able to specify a case-sensitive search by selecting the Case Sensitive checkbox.

The file search is performed in a separate thread via the java.lang.Runnable interface's run() method to allow the user the ability to interact with the utility. The run() method calls the updateUI() method with a true parameter to disable several user interface components prior to the search. The run() method then calls the checkForFile() method, which recur-

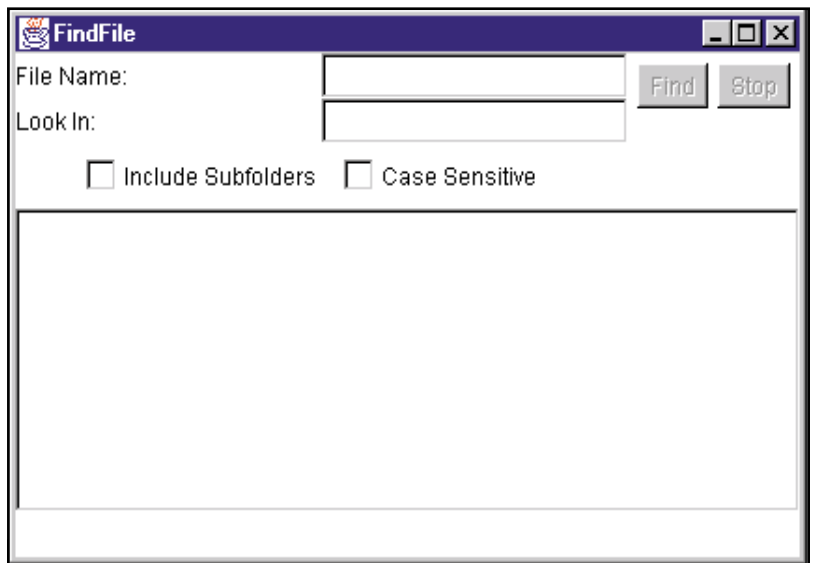


FIGURE 1 The FindFile utility

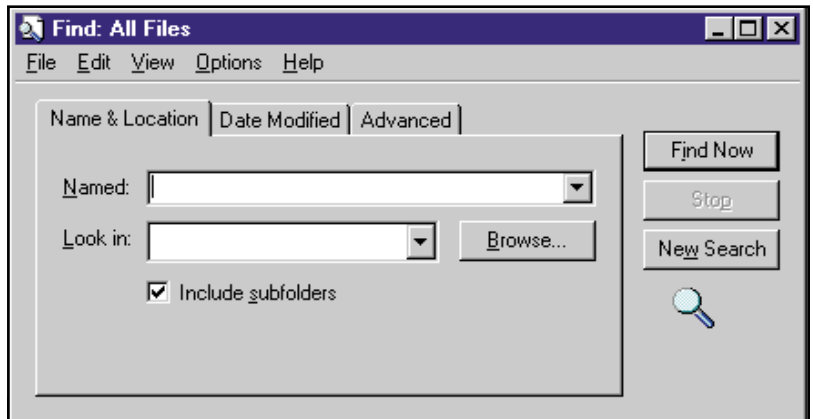


FIGURE 2 The Win32 Find utility

Segue Software

www.segue.com

JVM/COMPILER VERSION (compiled without optimization)	FINDFILE UTILITY	WIN32 FIND UTILITY
JDK 1.1.8	24.6 seconds	15.2 seconds
JDK 1.2.2	16.4 seconds	15.2 seconds
JDK 1.3.0 Release Candidate 2	15.8 seconds	15.2 seconds

FIGURE 3 FindFile utility performance comparison

sively cycles through any directories in the list only if the Include Subfolders checkbox is selected. The utility supports wild card character (i.e., * and ?) file searches through the use of a Java regular expression shareware package called pat, written by Steven R. Brandt and available at www.javaregex.com. Because the package uses a wild card character syntax slightly different from the format used for Win32 and UNIX operating systems, the entered filename is first converted to a format the package understands via the `convert()` method. If a file matching the specified filename is found, it's added to a `java.awt.List` component along with the file's fully qualified path. As the search progresses, the names of the directories searched are displayed in a `java.awt.Label` component located just below the search results. When the search is completed, the `updateUI()` method is called again, this time with a false parameter to enable the UI components.

AUTHOR BIO

Pat Paternostro is an associate partner with the Tri-Com Consulting Group in Rocky Hill, Connecticut. Tri-Com provides programming services for a wide variety of development tasks.

Threading Issues

A started thread normally stops when its `run()` method executes to completion. In the case of this utility, however, a Stop button is provided to allow the user to cancel the file search at any time. Prior to JDK 1.2.x, a thread was prematurely terminated by calling the `java.lang.Thread` class's `stop()` method. However, because of its unsafe behavior that method has since been deprecated (see the JDK 1.2.x javadoc HTML help file for the `java.lang.Thread` class's `stop()` method to view detailed information concerning the method's deprecation). Rather than use a deprecated, unsafe method I chose an alternative mechanism to terminate the file search prematurely. When the utility's Stop button is clicked, the boolean instance variable, `searching`, is set to false, causing the file search to end normally as this variable is interrogated in the `checkForFile()` method's for loop.

Performance

Typically, what is gained in portability with a Java program is usually lost in performance. However, with the advent of the HotSpot compiler this is becoming less of an issue. I compiled and tested the FindFile utility using the three major JDK releases (JDK 1.1.8, 1.2.2, and 1.3.0 RC2) on a 450 MHz Pentium III workstation with 96 MB of RAM running WinNT. The timings represent the average number of seconds both utilities took to search for the same file five times in succession. The results are listed in Figure 3. Your timings will vary based on the JVM and operating system in use, the operating system activity and the speed of your workstation's processor.

Summary

The FindFile utility detailed here provides a consistent user interface and consistent functionality when used in heterogeneous environments. The utility is quite functional as it stands; however, more features can easily be added if desired. ☪

ppaternostro@tricomgroup.com

InestSoft Technology

www.inetsoftcorp.com

HotDispatch.com

www.hotdispatch.com

JAVA MESSAGING SERVICE USING THE WITH ENTERPRISE JAVABEANS

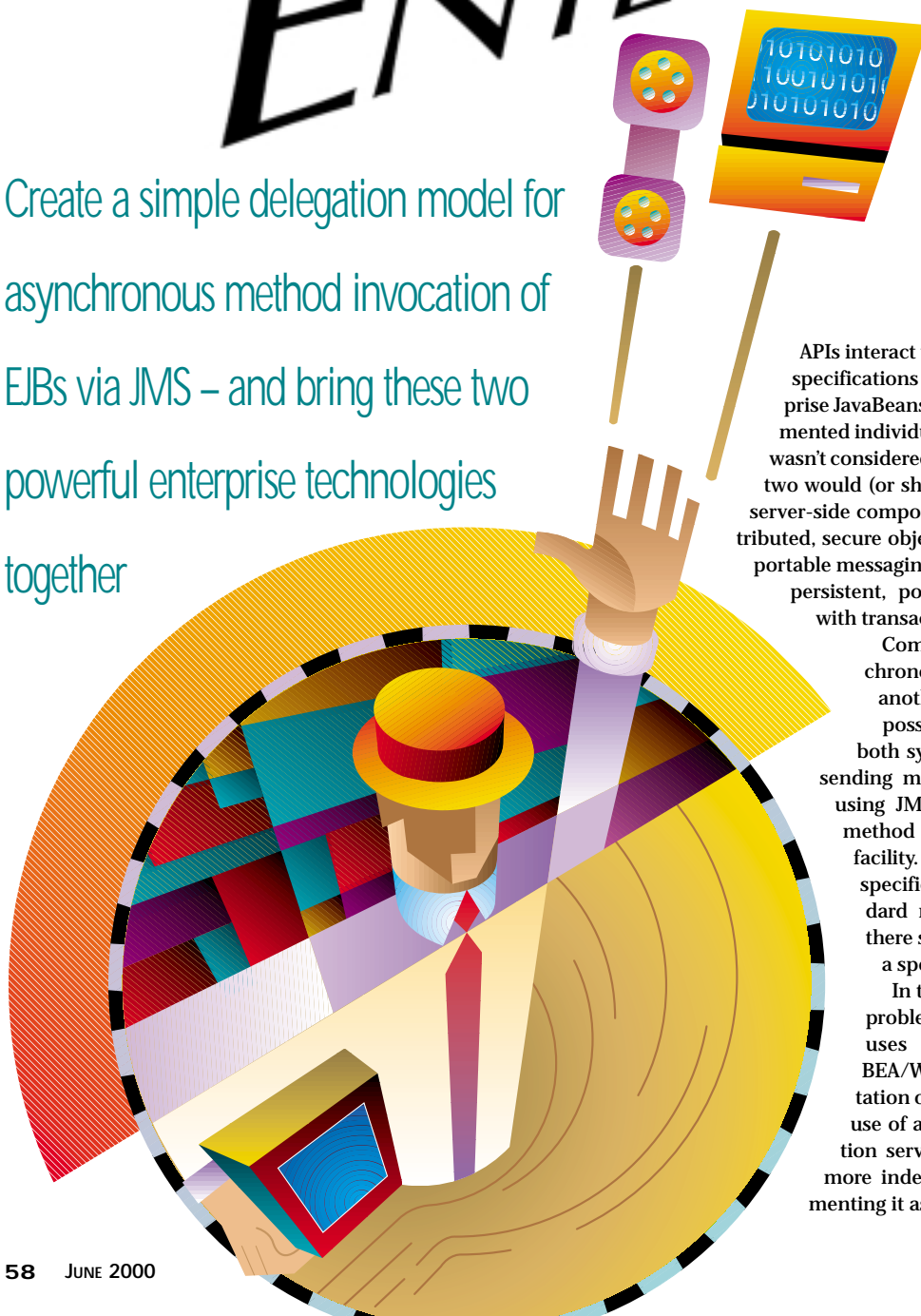
WRITTEN BY SCOTT GRANT

Create a simple delegation model for asynchronous method invocation of EJBs via JMS – and bring these two powerful enterprise technologies together

One unfortunate aspect of the many enterprise APIs and specifications that Sun has released over the last few years has been the lack of information about how some of these APIs interact with one another. In particular, two very useful specifications – the Java Messaging Service (JMS) and Enterprise JavaBeans (EJB) – have been released and already implemented individually by many application server vendors. What wasn't considered in this process, at least at this stage, is how the two would (or should) work together. Enterprise JavaBeans is a server-side component architecture, which provides remote, distributed, secure objects with built-in transaction support. JMS is a portable messaging service architecture and API that provides for persistent, point-to-point or publish/subscribe messaging with transaction support.

Communication with, or between, EJBs is a synchronous mechanism of one client method (perhaps another EJB) invoking a method on an EJB, with a possible return value. JMS, on the other hand, has both synchronous and asynchronous capabilities for sending messages between clients. It would seem that using JMS to pass asynchronous messages (or even method invocations) between EJBs would be a useful facility. Unfortunately, Sun has yet to tie these two specifications together and there is currently no standard mechanism to use JMS with EJBs (although there seems to be the promise of Sun delivering such a spec in the not-too-distant future).

In this article I'll show you how to circumvent this problem with an adapter-style delegation model that uses some of the inherent features of the BEA/WebLogic application server and its implementation of both JMS and EJBs. This model, while making use of a somewhat proprietary feature of the application server, can easily be extended if so desired to a more independent and portable mechanism by implementing it as a stand-alone Java service.



Intuitive Systems

www.optimizeit.com

If you want to test the example code included with this article, I would recommend that you first obtain and install the latest release of the BEA/WebLogic application server (this article is based on version 4.5.1). You can download a free 30-day trial copy from BEA at www.bea.com.

The Java Messaging Service

The JMS is a set of interfaces implemented by vendors of message-oriented middleware (MOM), application servers and database servers that wish to support messaging within their products. JMS provides a simple, common API for client applications to implement code that uses portable messaging, against potentially any given number of underlying messaging systems. (Because JMS is designed to be portable, it's important to realize that as a result, if you're familiar with any given MOM product, it isn't certain that JMS will support every aspect of that product.)

The primary concept in JMS is that of *Destinations*. A Destination is nothing more than an association for message producers and message consumers. Destinations break down into two types, Topics or Queues. For the purposes of this article we'll discuss only Queues, which implement point-to-point messaging. (Using a Topic to support the implementation described in this article should be a fairly easy substitution, however.) Both Queues and Topics support persistence. An incoming message will be stored in a persistent Queue until a QueueReceiver connects to it and receives the message synchronously through a receive() call, or passes it to a registered MessageListener. This latter mechanism provides an asynchronous message delivery model. JMS also provides support for transactions in a very basic form through the standard Connection/Session creation mechanisms. Alternately, JMS provides an XA implementation that is by default transacted and will participate in the context of a distributed transaction. A full description of transaction support is beyond the scope of this article. If you're unfamiliar with the JMS, I recommend my tutorial "Using the Java Mes-

saging Service with BEA/WebLogic" published in the January 2000 issue (*JDJ*, Vol. 5, issue 1).

Enterprise JavaBeans

EJBs are remote distributed server-side components that have built-in support for transactions and security. An EJB is a self-contained component that runs within the confines of an EJB container. The container, and its underlying server, are typically supplied by an application server provider. It's the duty of the container to manage any EJBs running within it by taking responsibility for:

- *Managing the state of an EJB*
- *Instantiating, pooling, removing and activating the EJB*
- *Making callbacks to the EJB to tell it to load or store its state* (with container-managed persistence, the container can even manage the loading and storing of the EJB's state)
- *Thread safety* (by default EJBs are single-threaded, and mustn't create their own threads)

Because EJBs are designed with inherent support for transactions, the container also manages the coordination of the EJB with an underlying Transaction Manager (the transaction may be a distributed transaction that in turn is coordinated with various resource managers for access to one or more databases).

Transaction management is therefore fairly transparent to EJB developers, requiring them to set a couple of DeploymentDescriptor attributes to specify how the EJB will participate in the context of a transaction (or even not at all). The DeploymentDescriptor allows the EJB to set certain parameters at the time it's deployed so the application server can change aspects of how the EJB is handled by the container (such as how it's pooled, how it participates in a transaction or its security attributes) without changing the EJB's code, or having to recompile it.

EJBs also provide a security model that can control access at the method level for any specific EJB. In this way you could have an EJB with

Listing 1

```
package jdj.article.jmsejb;

import javax.naming.*;
import javax.jms.*; // Import the javax.jms package
import java.util.Hashtable;

* JmsQueueManager.java
*
* An abstract base class that implements all the basic
* functionality for creating a JMS QueueReceiver or
* QueueSender. Includes all the necessary code for obtaining
* a JNDI naming Context, looking up the Queue and retrieving
* it, setting up the JMS Connection and Session, and creating
* the actual QueueReceiver and/or QueueSubscriber.
*
* This is extended by the ReceiverStartup and the
* JMSEjbClient classes, but it could also be used in a "has
* a" relationship by creating an implementation class and
* then contained as an attribute of our classes through a
* reference.
*
* @author Scott Grant
* @version 1.0 - 4/10/00
*
public abstract class JmsQueueManager implements IJmsEjbConstants
{
    protected QueueConnectionFactory queueFx; // Our factory
    protected QueueConnection conn; // Our connection
    protected Queue queue; // Our queue
    protected Queue tempQueue; // Our temporary queue
    protected QueueSession session; // Our session
    protected QueueReceiver receiver; // Our receiver
    protected QueueSender sender; // Our sender

    protected String jndiFactory;
    protected String url;
```

```
    protected String jmsFactory;
    protected String queueName;
    protected String principal;
    protected String credential;

    protected Context ctx; // our initial JNDI context

    * Obtain an initial context from Weblogic through JNDI (this
    * will be used to access the JMS factory and topic/queue
    * information).

    public JmsQueueManager(String jndiFactory, String url,
        String queueName, String jmsFactory,
        String principal, String credential)
    {
        this.jndiFactory = jndiFactory;
        this.url = url;
        this.queueName = queueName;
        this.jmsFactory = jmsFactory;
        this.principal = principal;
        this.credential = credential;
    }

    protected void finalize() throws Throwable
    {
        closeJMS();
    }

    protected Context getInitialContext()
    {
        // Try and create a new initial context using our
        // properties...
        Hashtable env = new Hashtable();
        env.put(Context.INITIAL_CONTEXT_FACTORY, jndiFactory);
        env.put(Context.PROVIDER_URL, url);
        env.put(Context.SECURITY_AUTHENTICATION, "simple");
        env.put(Context.SECURITY_PRINCIPAL, principal);
        env.put(Context.SECURITY_CREDENTIALS, credential);
```

WebGain

www.webgain.com

five methods, four of which were available to all users and one of which was restricted to a manager only, or an administrator – constituting very fine-grained security access control for an EJB.

One final aspect of EJBs is that by default they're distributed components. In this sense they're remote and are accessed in the same way RMI objects are. In fact, the various EJB interfaces (EJBHome, EJBObject) extend the `java.rmi.Remote` interface and methods of the EJBObject-derived remote interface throw `java.rmi.RemoteException`. EJBs are typically registered with JNDI (WebLogic does this through the `weblogic.properties` file at startup). Access to the EJB is a JNDI lookup operation for an EJB's home interface, which then acts as a type of factory for creation or location of the actual EJB itself.

One important thing to remember with EJBs is that you're always dealing with a "remote" stub class, you're never accessing the actual EJB directly (that is, you're never directly accessing the class that actually implements the functionality provided by the EJB – usually referred to as the "bean" class). Access to the actual functionality of the EJB is through calls to the stub class (which implements the EJB's remote interface). The stub communicates with the bean class through the container. The container controls access to the bean class and manages calls to the EJB, usually within a transaction context. It is this aspect of EJBs, though – their distributed, remote nature – that leads us to the problem of using EJBs with the JMS.

The Problem: Using JMS with EJBs

The problem in trying to use JMS with EJBs derives from the way you access an EJB to perform a method invocation on the bean. Specifically, the problem concerns the fact that an EJB doesn't behave like a standard "daemon service" or remote object in the sense that it isn't continuously up and running and waiting for access. EJBs are available during an access, but at any point after the access – after a specific timeout period or when the last reference to a bean is released – it can be passivated and put back into an available pool of beans by the container. Remember that

you never have access to the bean itself, but only to a client-side stub class that implements the EJB's remote interface. A reference to the stub doesn't guarantee that the bean is active on the server or available on it. (The bean may have been passivated due to a timeout, for example, or the object pool might be at its maximum.) The key point here is that the EJB isn't guaranteed to be constantly running and available simply because your code might have a reference to the EJB's remote interface.

On the messaging side, JMS provides a mechanism to send and receive messages to and from a Destination. Let's take the example of one specific Destination type, a Queue. You create a `QueueConnection` and `QueueSession`, and from the `QueueSession` you ultimately create a `QueueSender` or `QueueReceiver`. The `QueueSender` is used to send a JMS message to a Queue. Any `QueueReceiver` connected to the same Queue can retrieve incoming messages on the Queue synchronously or asynchronously. For a message to be delivered asynchronously, a `QueueReceiver` must have a registered `MessageListener` to which it can pass an incoming asynchronous message. An object that wishes to be notified of asynchronous messages must then implement the `MessageListener` interface and its single method `onMessage()`. Once this object is registered with the `QueueReceiver` for a specific Queue, any incoming message on that Queue will be passed to the `onMessage()` implementation of that object as a callback from JMS.

Sending a JMS message to a Queue is a simple matter for an EJB. When a method on an EJB is invoked, the bean can simply create the appropriate JMS Queue objects, retrieve the Queue from JNDI and ultimately send a specific JMS message to the Queue. But what happens if we want our EJB to receive a JMS message? At first glance it might seem that you could simply implement the `MessageListener` interface in your EJB's bean class, implement the `onMessage()` method and register this with a `QueueReceiver`. But this implementation violates one of the major rules of EJBs and is therefore illegal and potentially disastrous. Why?

It all comes back to two major points discussed above: an EJB bean class should never be accessed directly, but only through its remote interface; and the Container manages the availability of an EJB.

```
try
{
    // Try and create a new initial context using
    // our properties...
    Context context = new InitialContext(env);
    return context;
}
catch(NamingException e)
{
    System.out.println("getInitialContext: Could not
    obtain initial naming context");
    e.printStackTrace();
    return null;
}
catch(Exception e)
{
    System.out.println("getInitialContext: Unknown
    exception trying to obtain initial naming context");
    e.printStackTrace();
    return null;
}
}

* Initialize JMS - create initial JMS resources,
* start connection, etc.

protected void initializeJMS(String type, MessageListener
listener, boolean transacted)
{
    try
    {
        if (ctx == null)
            ctx = getInitialContext();

        if (ctx != null)
        {
            // Get the default queue connection
```

```
// factory...Destinations (Queues and
// Topics) and ConnectionFactory objects are
// administered objects - you retrieve them
// from the Weblogic Application Server via JNDI.
queueFx = (QueueConnectionFactory)
ctx.lookup("jmsFactory");

// Get a QueueConnection from the QueueCon-
// nectionFactory
conn = queueFx.createQueueConnection();

// Get a QueueSession - auto-acknowledge -
// and use the parameter to
// determine if it is transacted or not...
session = conn.createQueueSession(transacted,
Session.AUTO_ACKNOWLEDGE);

// Get the Queue from JNDI...Queue is an
// administered object
queue = (Queue) ctx.lookup(queueName);

if ((type.equals(RECEIVER) ||
type.equals(SENDER_RECEIVER)) &&
listener != null)
{
    // Create a Receiver for the Queue...
receiver = session.createReceiver(queue);

    // Set the listener (this class)
receiver.setMessageListener(listener);
}

if (type.equals(SENDER) ||
type.equals(SENDER_RECEIVER))
{
    sender = session.createSender(queue);
}
```

Buzzeo

www.buzzeo.com

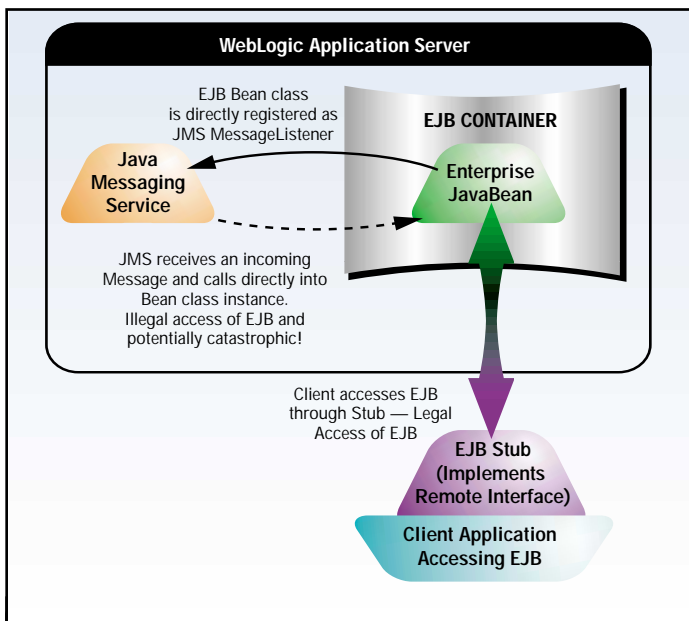


FIGURE 1 Illegal use of JMS with EJB

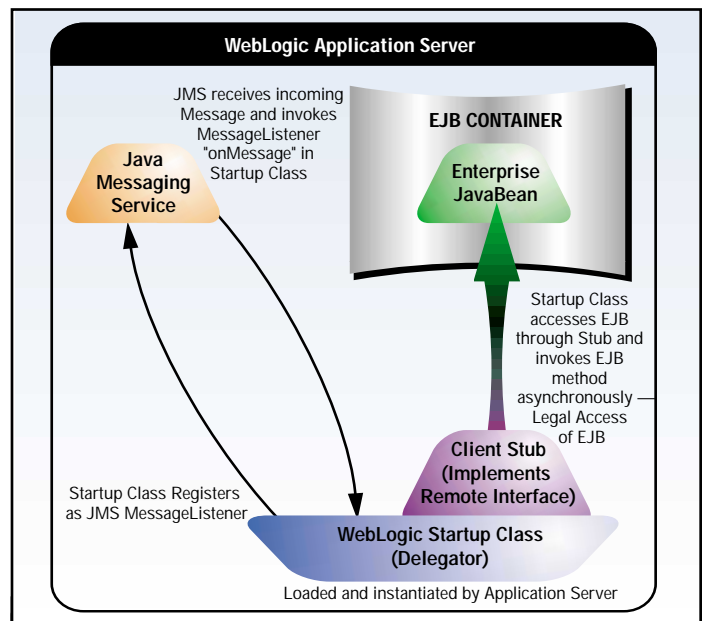


FIGURE 2 Legal use of JMS with EJB via delegation model

If you implement the MessageListener in the bean class, JMS is essentially calling directly into the EJB and not through the remote interface. Thus JMS is bypassing the container and directly making an invocation on the bean class. Because the container is managing not only the EJB but also transactions (as well as thread safety), it would be a potentially catastrophic operation if JMS were suddenly to call into your bean class directly while it's in the middle of a transaction or other operation. In addition, unless the EJB bean class was actually available, it wouldn't be possible to register the bean class with the QueueReceiver until the bean was in an instantiated state. (Also, depending on how such a sce-

nario was coded, JMS might be calling into a bean instance that was in the container's object pool, as well as calling into activated instances that were in communication with a client.) Figure 1 illustrates this kind of illegal use.

The Solution: WebLogic Startup Classes

The way to solve this problem is to use a delegation model. In the model I'm going to present we'll implement a delegator class (a kind of adapter) for receiving and forwarding JMS messages to an EJB. We can

```

// Start connection...
conn.start();
}
else
{
    System.out.println("initializeJMS: InitialContext was null");
    return;
}
}
catch(NamingException e)
{
    System.out.println("initializeJMS: NamingException was thrown");
    e.printStackTrace();
}
catch(JMSSecurityException e)
{
    System.out.println("initializeJMS: JMSSecurityException was thrown");
    e.printStackTrace();
}
catch(JMSEException e)
{
    System.out.println("initializeJMS: JMSEException was thrown");
    e.printStackTrace();
}
catch(Exception e)
{
    System.out.println("initializeJMS: Unknown exception trying to obtain initialize JMS");
    e.printStackTrace();
}
}

* Close down JMS - stop the connection, etc.

```

```

protected void closeJMS()
{
    if (conn != null)
    {
        try
        {
            conn.stop();
            if (sender != null)
                sender.close();

            if (receiver != null)
                receiver.close();

            if (session != null)
                session.close();

            conn.close();
        }
        catch(JMSEException e)
        {
            System.out.println("closeJMS: JMSEException thrown while trying to close connections");
            e.printStackTrace();
        }
        catch(Exception e)
        {
            System.out.println("closeJMS: An unknown exception occurred while trying to close connections");
            e.printStackTrace();
        }
    }
}

* sendMessage
* Send a JMS Message.

protected void sendMessage(Message msg)

```


Tidestone Technologies

www.tidestone.com

implement this delegation class using a proprietary feature of the WebLogic application server called "startup classes." A startup class is simply a Java class that implements the `weblogic.common.T3StartupDef` interface from the WebLogic packages.

A startup class is registered in the `weblogic.properties` file and, as the name implies, will be loaded by the WebLogic server upon startup and run within the application server's VM instance. The `T3StartupDef` interface defines one method called `startup()`, which is passed a String identifier and a `Hashtable` that contains any parameters you wish to pass to your startup class – these parameters are set in the `weblogic.properties` file. The `startup()` method is the equivalent of a `main()` method, in a sense, and it will be called by the WebLogic server when your startup class is instantiated.

Next we need to create a special Queue (see the WebLogic documentation for details on setting up the Queue for the WebLogic application server) in the WebLogic server for our incoming JMS messages. We'll use this special Queue when sending messages intended for an EJB via the startup class delegator. In turn, the startup class is set up to listen on this same Queue for JMS messages intended for an EJB. We'll use this model for two types of EJB access. The first is a direct delegation model in which we'll pass the incoming JMS message on to an `onMessage()` method in our example EJB. The second is an asynchronous method invocation on an EJB via a JMS message (see Figure 2). Both forms actually use the same underlying code, as you will see.

Using the Delegation Model

Our startup class should contain code within its `startup()` method that initializes JMS and implements the `JMS MessageListener` interface and its `onMessage()` method. Listing 1 shows the code for our `JmsQueueManager` class – this class implements all of the functionality required to create a `JMS QueueConnection`, `QueueSession`, `QueueReceiver` and `QueueSender`. It also provides utility methods that retrieve a

Queue via JNDI and shuts down the JMS connection in its `finalize()` method.

The important method in `JmsQueueManager` is `initializeJMS()`. This initializes JMS, creates the `JMS QueueConnection`, `QueueSession`, retrieves the Queue through JNDI and creates the `QueueSession` and/or `QueueReceiver`.

We extend this base class in our WebLogic startup class. This gives our startup class all the JMS functionality it needs to manage a `JMS Connection`. Our startup class also implements the `JMS MessageListener` interface and its `onMessage()` method. Once JMS is successfully initialized through the base class code, any incoming JMS messages sent to our Queue will cause an invocation of the `onMessage()` method in the startup class. The `onMessage()` method, the core of the startup class, is where the actual delegation occurs. We do the delegation through the use of Java's reflection mechanism and by parsing out method names and parameters from the contents of a JMS message. The startup class is shown in Listing 2.

Asynchronous Method Invocation with JMS

The `onMessage()` method of the startup class extracts predefined parameters from an incoming JMS message. In the code listing this message is assumed to be a `JMS MapMessage` that can contain multiple name/value pairs. A value is stored in conjunction with a String "name." You look up the value by passing the String name to an appropriate method, based on the type of the value – for instance, `"getString(String name)"` will look for a String value based on "name." It uses the values of these parameters to obtain the `EJBHome` class, first. This is important since we need the `EJBHome` to either create or locate an EJB. Then, using the Java reflection mechanism, we utilize additional JMS message parameters to determine the type of parameters for two methods; the `EJBHome`'s `create()` method (this could also support an `EntityBean`'s `findByPrimaryKey()` method) and then the actual method to be invoked on the remote interface.

```
{
    try
    {
        msg.setJMSDeliveryMode(DeliveryMode.PERSISTENT);
        sender.send(msg); // default behavior
    }
    catch(JMSEException e)
    {
        System.out.println("initializeJMS: JMSEException trying to send message to queue");
    }
}

* createMapMessage
*
* Creates a MapMessage - delegates to
* the session version of this method.

protected MapMessage createMapMessage()
{
    try
    {
        return session.createMapMessage(); // default behavior
    }
    catch(JMSEException e)
    {
        System.out.println("createMapMessage: JMSEException trying to create MapMessage");
    }

    return null;
}

// You could implement all the additional "createXXXMes0
// sage()" delegator methods here, if you wanted to flesh
// out this class...
}
```

Listing 2

```
package jdj.article.jmsejb;

import javax.naming.*;
import javax.jms.*; // Import the javax.jms package
import javax.ejb.*;
import java.util.*;
import java.lang.reflect.*;
import java.rmi.RemoteException;

import weblogic.common.*;

* JmsEjbReceiver.java
*
* Implements Weblogic Start-Up class that receives JMS
* Messages on an incoming Queue using Weblogic's JMS
* implementation.
*
* @author Scott Grant
* @version 1.0 - 4/10/00

public class ReceiverStartup extends JmsQueueManager
    implements MessageListener,
        T3StartupDef
{
    // Note: These parameters could be passed in the Hashtable
    // arguments to the Weblogic Start-Up Class via the
    // "weblogic.properties" file...I have hard coded them here...
    public static final String JNDI_FACTORY =
        "weblogic.jndi.WLInitialContextFactory";
    public static final String URL = "t3://localhost:7001";
    public static final String JMS_FACTORY =
        "javax.jms.QueueConnectionFactory";
    public static final String QUEUE =
        "jdi.article.jmsejb.ejbMessageQueue";
    public static final String PRINCIPAL = "system";
    public static final String CREDENTIAL = "password";
}
```

Flashline

www.flashline.com

In the code listing I've hard-coded the String "names" for the MapMessage's values for clarity (these could be turned into "final static" constants). This shows how you can use a single JMS message to package all the information needed for the delegator so it can locate the EJBHome, use reflection to create the EJB and – if the remote interface stub class is successfully retrieved – invoke a method on the stub class passing in parameters from the MapMessage. Thus the delegation startup class is making asynchronous invocations against the EJB. The code for the ReceiverStartup class in Listing 2 shows how to do this.

This mechanism uses some support methods to obtain the Class of the parameters for both a create method and an EJB's remote interface method, and to determine and set the parameter's values. Once these have been created, the method is actually invoked through reflection.

Delegating the JMS Message

An important point to note is that we use the same block of code in onMessage() to invoke a method asynchronously on an EJB and to pass on a JMS message to an EJB.

The code in Listing 3 shows how we would do this. This is a simple client class that can be used to test our delegator. It creates two separate JMS messages. The first sets its parameter to type "message." Doing this instructs the ReceiverStartup onMessage() code to look for the method name in the remote interface stub, but to pass on the JMS message to the EJB's method (this method, of course, must take a parameter that is a JMS message). In this way the ReceiverStartup class performs what appears to be a simple pass through of the message to the EJB. The second message in the client code listing sends a message with multiple parameters that invokes a simple EJB method called testMethod() that takes an integer and String as parameters. Thus we have one block of code in the ReceiverStartup class that can be used for any asynchronous EJB invocation, including passing a JMS message on to the EJB itself (although you must add additional parameters to the message so that the EJB can be located).

Some additional features that could be added to the code would be dealing with return values using the JMSReplyTo property of a JMS message (not implemented in the code listings in this article – I leave you to investigate this for yourself). You could specify a "return value" Queue on which you could pass back a message with a unique identifier and the returned results from the method that was invoked. Also, in the code listings I use a JMS MapMessage that requires us to add the specific name/values that we'll need to locate the EJB and invoke a method on it. This could be enhanced by wrapping or extending the base JMS message classes to add methods to handle these unique name/values, while leaving the underlying JMS message alone.

Summary

There's little doubt that EJBs and JMS are both very useful and powerful enterprise technologies when used independently. Although Sun has yet to define how these two specifications should interact, the model presented here demonstrates one solution to the current problem developers face when trying to use these technologies together. It's my hope that the ideas presented in this article demonstrate the potential for using JMS and EJBs in the enterprise. I encourage you to experiment with the concepts presented and create your own EJB and JMS solutions. ☺

AUTHOR BIO

Scott Grant is chief architect and lead developer for CascadeWorks, Inc., a San Francisco-based ASP company providing B2B solutions. A Sun-certified Java developer with 15 years of diversified engineering experience, Scott previously pioneered Java enterprise e-service solutions at another Bay Area startup company.

sgrant@cascaadeworks.com

```
private T3ServicesDef serv; // Part of Weblogic Start-Up
class support

public ReceiverStartup()
{
    super(JNDI_FACTORY, URL, QUEUE, JMS_FACTORY, PRINCIPAL, CREDENTIAL);
}

// Weblogic Start-Up Class

public void setServices(T3ServicesDef s)
{
    serv = s;
}

* startup
*
* This is part of the Weblogic T3StartupDef
* interface
*
public String startup(String name, Hashtable args) throws
Exception
{
    ctx = getInitialContext();

    if (ctx != null)
        initializeJMS(RECEIVER, this, false); // Not
        using transactions
    else
        throw new NamingException("ReceiverStartup -
        initializeJMS: Naming Exception was thrown");

    return "ReceiverStartup listening...";
}

// JMS Items

* Implements the onMessage method of the MessageListener
* interface. This is the call back method used by JMS to
```

```
* pass us messages on the queue our which we're
* registered with...

public void onMessage(Message msg)
{
    System.out.println("ReceiverStartup: Received incoming
    JMS message...");

    try
    {
        String msgType = msg.getJMSType();
        if (msg instanceof MapMessage &&
            msgType.equals(JMSEJB_MESSAGE))
        {
            MapMessage mapMsg = (MapMessage)msg;
            String homeName = mapMsg.getString("HomeName");
            String createName = mapMsg.getString("CreateName");
            int createParams = mapMsg.getInt("CreateParams");
            String methodName = mapMsg.getString("MethodName");
            int methodParams = mapMsg.getInt("MethodParams");

            System.out.println("homeName: " + homeName);

            if (ctx == null)
                ctx = getInitialContext();

            if (ctx != null)
            {
                try
                {
                    // Get the class type and parameters
                    // for the EJBHome create method...
                    System.out.println("Getting create
                    class types...");
                    Class[] createTypes = get
                    ClassTypesFromMessage((MapMessage)msg, createParams, "Cre-
                    ateParam", "CreateParamType");
                    System.out.println("Getting create
                    arguments and types...");
                    Object[] createArgs = getArguments-
                    FromMessage((MapMessage)msg, createParams, "CreateParam",
```

Modis Solutions

www.idea.com

```

>CreateParamType");

        // Get the class type and param-
        // ters for the EJBObject (stub)
        // method to invoke...
        System.out.println("Getting method
class types...");
        Class[] methodTypes = get-
ClassTypesFromMessage((MapMessage)msg, methodParams, "Method-
Param", "MethodParamType");

        System.out.println("Getting method
arguments and types...");
        Object[] methodArgs = getArguments-
FromMessage((MapMessage)msg, methodParams, "MethodParam",
"MethodParamType");

        System.out.println("Looking up
object class name...");

        // Find our home class through JNDI...
        Object obj = ctx.lookup(homeName);
        Class homeClass = obj.getClass();

        // Find the create method...
        Method methodCreate =
homeClass.getMethod(createName, createTypes);

        System.out.println("Invoking create
method...");

        // Invoke the create method...
        Object ejb =
methodCreate.invoke(obj, createArgs);

        System.out.println("Invoking method
" + methodName + "...");

        // Find the EJB method in the JMS
        // Message...
        Method methodCall = ejb.get-
Class().getMethod(methodName, methodTypes);

        // Invoke the EJB method...
        methodCall.invoke(ejb, methodArgs);

    }
    catch(InvocationTargetException e)
    {
        System.out.println("ReceiverStart-
up: InvocationTargetException was thrown");
    }
    catch(NoSuchMethodException e)
    {
        System.out.println("ReceiverStart-
up: NoSuchMethodException was thrown");
    }
    catch(IllegalAccessException e)
    {
        System.out.println("ReceiverStart-
up: IllegalAccessException was thrown");
    }
    catch(Exception e)
    {
        System.out.println("ReceiverStart-
up: An exception was thrown");
    }
}
else
    System.out.println("ReceiverStartup:
onMessage: Failed to obtain initial naming context");
}
catch (JMSEException e)
{
    System.out.println("ReceiverStartup: onMessage:
JMSEException was thrown");
    e.printStackTrace();
}
}

* getClassTypesFromMessage
*

```

```

* Returns an array of Class types from a MapMessage
* which contains a list of parameters for an EJB method call.

public Class[] getClassTypesFromMessage(MapMessage msg,
int params,
String argName, String typeName) throws JMSEException
{
    Class[] types = new Class[0];

    if (params > 0)
    {
        types = new Class[params];

        for (int i = 0; i < params; i++)
        {
            String type = msg.getString(typeName + i);
            if (type.equals("String"))
            {
                types[i] = String.class;
            }
            else if (type.equals("int"))
            {
                types[i] = int.class;
            }
            else if (type.equals("long"))
            {
                types[i] = long.class;
            }
            else if (type.equals("float"))
            {
                types[i] = float.class;
            }
            else if (type.equals("double"))
            {
                types[i] = double.class;
            }
            else if (type.equals("short"))
            {
                types[i] = short.class;
            }
            else if (type.equals("byte"))
            {
                types[i] = byte.class;
            }
            else if (type.equals("object"))
            {
                // Extract any type of object...
                Object obj = msg.getObject(argName + i);
                types[i] = obj.getClass();
                System.out.println("Object instance: "
+ obj.getClass().getName());
            }
            else if (type.equals("message"))
            {
                types[i] = Message.class;
            }
            else
                System.out.println("Unknown parameter
type");
        }
    }

    return types;
}

* getArgumentsFromMessage
*
* Returns an array of Object's from a JMS MapMessage
* that are extracted as parameters for an EJB method
* call.

public Object[] getArgumentsFromMessage(MapMessage msg,
int params,
String argName, String argType) throws JMSEException
{
    Object[] args = new Object[0];

    if (params > 0)
    {
        args = new Object[params];

        for (int i = 0; i < params; i++)

```

Persistence

www.persistence.com

```

    {
        String type = msg.getString(argType + i);
        if (type.equals("String"))
        {
            String temp = msg.getString(argName + i);
            args[i] = temp;
        }
        else if (type.equals("int"))
        {
            int temp = msg.getInt(argName + i);
            args[i] = new Integer(temp);
        }
        else if (type.equals("long"))
        {
            long temp = msg.getLong(argName + i);
            args[i] = new Long(temp);
        }
        else if (type.equals("float"))
        {
            float temp = msg.getFloat(argName + i);
            args[i] = new Float(temp);
        }
        else if (type.equals("double"))
        {
            double temp = msg.getDouble(argName + i);
            args[i] = new Double(temp);
        }
        else if (type.equals("short"))
        {
            short temp = msg.getShort(argName + i);
            args[i] = new Short(temp);
        }
        else if (type.equals("byte"))
        {
            byte temp = msg.getBytes(argName + i);
            args[i] = new Float(temp);
        }
        else if (type.equals("object"))
        {
            Object obj = msg.getObject(argName + i);
            args[i] = obj;
        }
        else if (type.equals("message"))
        {
            args[i] = msg;
        }
        // You could add other types here...Object,etc...
        else
            System.out.println("Unknown parameter type");
    }
}

return args;
}
}

```

Listing 3

```

package jdj.article.jmsejb;

import javax.jms.*;
import javax.naming.*;

* JmsEjbClient
*
* This is our client class. It extends the base JmsQueueMan-
* ager to handle the creation of the JMS connection, session,
* and QueueSender.
*
* It creates two JMS MapMessages. The first is sent to the
* EJB's "onMessage" method through simple delegation. The
* second message is actually used to call a method on the EJB
* asynchronously via a JMS message - both cases are handled
* by the ReceiverStartup Weblogic startup class.
*
* @author Scott Grant
* @version 1.0 - 4/10/00

public class JmsEjbClient extends JmsQueueManager
{
    public static final String JNDI_FACTORY =
        "weblogic.jndi.WLInitialContextFactory";
    public static final String URL = "t3://localhost:7001";

```

```

public static final String JMS_FACTORY =
    "javax.jms.QueueConnectionFactory";
public static final String QUEUE =
    "jdj.article.jmsejb.ejbMessageQueue";
public static final String PRINCIPAL = "system";
public static final String CREDENTIAL = "password";

public JmsEjbClient()
{
    super(JNDI_FACTORY, URL, QUEUE, JMS_FACTORY, PRINCI-
        PAL, CREDENTIAL);
}

public static void main(String[] args)
{
    JmsEjbClient client = new JmsEjbClient();
    client.initializeJMS(SENDER, null, false);

    MapMessage msg = client.createMapMessage();
    try
    {
        // Create a MapMessage and send it to the Queue
        msg.setJMSType(JMSEJB_MESSAGE);
        msg.setString("HomeName",
            "jdj.article.jmsejb.JmsEjbExampleHome");
        msg.setString("CreateName", "create");
        msg.setInt("CreateParams", 0);

        // We are sending this method to the "onMessage"
        // method of the EJB, so we set one parameter
        // which is of type "message". This is a special
        // case and the ReceiverStartup will interpret
        // this as delegation - it will pass the actual
        // JMS Message itself, on to this method (onMes-
        // sage) of the EJB as the parameter.
        msg.setString("MethodName", "onMessage");
        msg.setInt("MethodParams", 1);
        msg.setString("MethodParamType0", "message");
        msg.setString("StringMessage", "This is my test
            message string sent to the EJB onMessage method...");

        client.sendMessage(msg);

        // Now send a method invocation with Params...

        msg = client.createMapMessage();
        msg.setJMSType(JMSEJB_MESSAGE);
        msg.setString("HomeName",
            "jdj.article.jmsejb.JmsEjbExampleHome");
        msg.setString("CreateName", "create");
        msg.setInt("CreateParams", 0);

        // We are sending this message to ReceiverStart-
        // up as a method
        // invocation. So we set the method name, and
        // the parameter types, and values. The
        // ReceiverStartup class will use Java's reflec-
        // tion mechanism to find this method on the
        // EJB, and invoke it, passing
        // in the parameters as we've specified them.
        // This demonstrates using a Weblogic startup
        // class as a delegator for asynchronous
        // method invocation on an EJB...
        msg.setString("MethodName", "testMethod");
        msg.setInt("MethodParams", 2);
        msg.setString("MethodParamType0", "int");
        msg.setInt("MethodParam0", 1);
        msg.setString("MethodParamType1", "String");
        msg.setString("MethodParam1", "This is my test
            message string sent to the EJB testMethod as method parameter...");

        client.sendMessage(msg);
    }
    catch(JMSEException e)
    {
        System.out.println("JmsEjbClient: JMSEException
            was thrown");
    }
}
}

```



Elixir Technology

www.elixirtech.com/download

Building Multimedia Repositories

Using Java and JDBC to store and retrieve multimedia information from a relational database



WRITTEN BY
SAMIR SHAH

Information repositories are essential. They allow data to be shared within or outside an organization, bringing us closer to the reality of the paperless office.

With the toolset shown in Table 1, you can build an enterprise-class, scalable Web-enabled repository that fully incorporates various forms of media. Document files, photographs, video clips and sound files can easily be included in the repository using Java and Oracle8i's LOB (Large Objects) data types.

- JDEVELOPER 2.0 WITH JDK 1.1.7
- ORACLE 8.1.5 JDBC THIN DRIVER
- ORACLE8i (8.1.5) ENTERPRISE SERVER
- JAVA WEB SERVER 2.0
- ORACLE INTERMEDIA 8.1.5.
- PLATFORM: WINDOWS 2000 SERVER

TABLE 1 Toolset for this article

In this article I'm going to focus on how you build a repository to store and search documents such as Microsoft Word, and HTML and XML files stored in a LOB column of a database table. The example used here populates the repository with Microsoft Word résumés, indexes it using Oracle Intermedia and reads it using Java streams from a servlet (see Figure 1).

Benefits of Java and Oracle8i

Building repositories using Java and Oracle8i has several benefits. The docu-

ments inherently take advantage of the transaction management and ACID (atomicity, concurrency, integrity and durability) properties of the relational database, which means that changes to an internal LOB can be committed or rolled back. Moreover, associated applications can seamlessly take advantage of database features such as backup and recovery. This makes things easier for the system administrators, who no longer have to perform separate database and file system backups for relational information and documents. All data housed in the database, whether structured (relational) or unstructured (document files), can be written, searched and accessed using a single industry standard interface – SQL. These SQL statements can be executed from Java using JDBC (Java Database Connectivity).

Working with Large Objects

Oracle8i supports several types of LOB columns. One type, a BLOB (Binary Large Object), can house binary information such as audio, video, images and documents internally within the database. Each row can store up to 4 gigabytes of data. I used the BLOB data type to store the Microsoft Word résumés in my example.

The Oracle database stores a locator inline with the data. The locator is a pointer to the actual location of the data (LOB value). The LOB data can be stored in the same table or a separate one. The advantage of the locator is that the database doesn't have to scan the LOB data each time it reads multiple rows because only the LOB's locator value is read; the actual LOB data is read only when required.

When working with Java and LOBs, first execute the SELECT statement to get the LOB locator, then read or write LOBs using JDBC. (Oracle JDBC driver's type extension classes from oracle.sql package is used to read and write from an Oracle database.) The actual LOB data is materialized as a Java stream from the database, with the locator representing the data in the table. The following code reads the résumé of an employee whose employee number, 7900, is stored in a LOB column called résumé in the sam_emp table.

```
Statement st = cn.createStatement();
ResultSet rs = st.executeQuery
("Select resume from sam_emp where
empno=7900");
rs.next();
oracle.sql.BLOB blob=((OracleResult-
Set)rs).getBLOB(1);
InputStream is=blob.getBinary-
Stream();
```

Populating the Repository

The documents can be written to LOB columns using Java, PL/SQL or a bulk utility called Oracle SQL*Loader. To insert a new row, do the following:

1. Execute the SQL insert statement with an empty BLOB.
2. Query the same row to get the locator object. Use this locator to write your document to the LOB column. *Note:* Java streams are employed to write the documents to the LOB column.

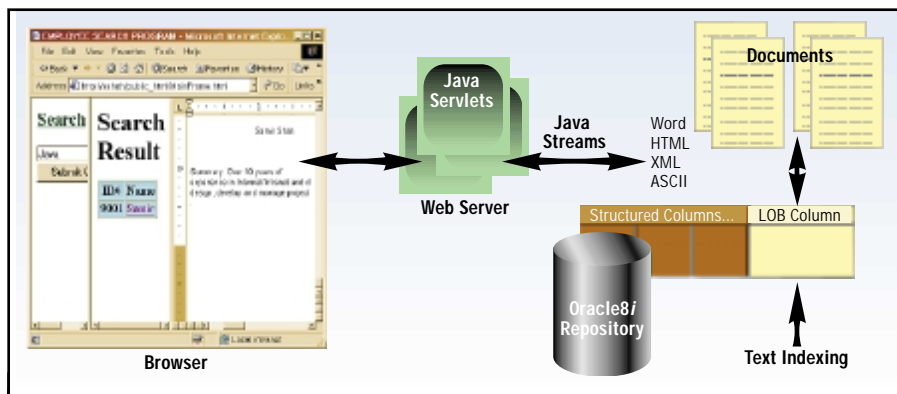


FIGURE 1 Java and Oracle8i repository

Software AG

www.softwareag.com/bolero

3. Create the Java output stream using the `getBinaryOutputStream()` method of this object to write your document or any binary information to that column.

For example, to insert information in the `sam_emp` table about a new employee whose employee number is 9001, first insert all the structured information along with an empty BLOB using JDBC. Next, select the LOB column, `résumé`, from the same row to get the `oracle.sql.BLOB` object (the locator). Finally, create the Java output stream from this object:

```
st.execute("INSERT INTO
sam_emp(empno, resume)
VALUES(9001,empty_blob())");
ResultSet rs = st.executeQuery(
    "Select résumé from sam_emp
where empno=9001 for update");
rs.next();
oracle.sql.BLOB blob = ((OracleRe-
sultSet)rs).getBLOB(1);
OutputStream os = blob.getBinaryOut-
putStream();
```

AUTHOR BIO

Samir Shah, a former Oracle employee and a certified Oracle professional, is a manager of database and Web technologies at Wall Street Systems. He has over 10 years' industry experience.

Optionally, you may use the `java.awt.FileDialog` class and `java.io` package to dynamically select and read a file from your PC. Then load it to a LOB column using the preceding code.

The way you search and retrieve documents is independent of how you load the documents. For example, you can store the documents using PL/SQL or SQL*Loader, then search and retrieve using Java servlets. Using PL/SQL, Listing 1 loads an employee's `résumé`, saved as a Microsoft Word file, to the `résumé` column of the `sam_emp` table.

Searching the Repository

The documents stored in the LOB columns can be indexed using Oracle Intermedia, which provides advance search capabilities such as fuzzy, stemming, proxy, phrases and more. It can also generate thematic searches and gist. The documents can be indexed via the "Create Index" database command.

Refer to Listing 2 to see how the index is built on the `résumé` column of the `sam_emp` table. Once the index is created, the Java applications can search the repository by simply submitting SELECT statements.

The `MyServletCtx` servlet in Listing 3 searches the term passed to it as a parameter in the `résumé` column of the `sam_emp` table. The servlet returns the rows matching the search criteria in HTML table format. The employee names in the HTML table are hyperlinked to another servlet, `MyServlet`,

which reads the entire `résumé` from the database in its original format.

Retrieving from the Repository

Document retrieval using Java is similar to writing documents to the repository. The "Working with Large Objects" section earlier in this article describes how to read LOBs from the database. The `MyServlet` in Listing 4 reads a Microsoft Word `résumé` from the `sam_emp` table, sets the content type, then streams it out to the browser using an output stream.

Summary

In this article I've shown how you store, search and retrieve Word documents using LOB data types and Java.

You can also use the Oracle8i database to store, index, parse and transform XML documents. Storing XML documents in the database removes the need to administer and manage multiple repositories for relational and XML data. The Oracle8i's JVM makes it possible to run a Java XML parser in the database. Using the parser, you can parse and transform the XML files inside the database before outputting it to an application server.

sssshah@yahoo.com

Listing 1: Inserting Word document in BLOB column using PL/SQL

The following code (Steps 2-5) inserts `MyResume.doc` in the `résumé` column of `sam_emp` table.

Step 1: Create a directory object in Oracle.

Here's how to create a directory object called `MY_FILES` that represents `C:\MY_DATA` directory. You must have the create directory privilege in Oracle.

```
create or replace directory
MY_FILES as 'C:\MY_DATA';
```

Step 2: Insert a row with an empty BLOB in your table and return the locator.

Step 3: Point to the Word file to be loaded from the directory created in Step 1 using `bfile` data type.

Step 4: Open the file and use the locator from Step 2 to insert the file.

Step 5: Close the file and commit the transaction.

```
declare
f_lob bfile;
b_lob blob;
begin
insert into sam_emp(empno,ename, resume)
values ( 9001, 'Samir',empty_blob() )
return résumé into b_lob;

f_lob := bfilename('MY_FILES', 'MyResume.doc' );
dbms_lob.fileopen(f_lob, dbms_lob.file_readonly);
dbms_lob.loadfromfile
( b_lob, f_lob, dbms_lob.getlength(f_lob) );
dbms_lob.fileclose(f_lob);

commit;
end;
/
```

Listing 2: Building index to search documents

The steps listed below index all the Microsoft Word-formatted `résumés` stored in the `résumé` column to the `sam_emp` table. The `résumés` can then be searched using SQL.

Step 1: Add a primary key to your table if it does not exist. To make `empno` primary key of the `sam_emp` table execute the following command.

```
alter table sam_emp add constraint
pk_sam_emp primary key(empno);
```

Step 2: Get the privileges (`ctxapp` role) to create text indexes from administrators.

Step 3: Create the index with the appropriate filter object. Filters determine how to extract text for document indexing from the word processor, formatted documents as well as plain text. See Oracle8i intermedia Text for complete list of filters.

```
create index ctx_doc_idx on sam_emp(résumé)
indextype is ctxsys.context parameters
('filter CTXSYS.INSO_FILTER');
```

Listing 3: Searching documents using SQL and JDBC

```
package package1;
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.sql.*;

/**
```

This servlet searches documents stored in an Oracle8i database repository using SQL and JDBC. The hit list is displayed in an HTML table with hyperlinks. JDK 1.1.7 and an Oracle Thin JDBC 1.22 compliant driver used.

```
*
* @author Samir Shah
```

Unify Corporation

www.servletexec.com

```

* @version 1.0
**/
public class MyServletCtx extends HttpServlet{
    Connection cn;

    public void init(ServletConfig parml)
    throws ServletException {
        super.init( parml);
        try{
            DriverManager.registerDriver(
            (new oracle.jdbc.driver.OracleDriver()));
            cn =DriverManager.getConnection
            ("jdbc:oracle:thin:@sshah:1521:o8i",
            "scott", "tiger");
        }
        catch (SQLException se){se.printStackTrace();}
    }

    public void doGet(HttpServletRequest req,
    HttpServletResponse res) throws IOException{

        doPost(req,res);
    }

    public void doPost(HttpServletRequest req,
    HttpServletResponse res) throws IOException{

        PrintWriter out = res.getWriter();
        res.setContentType("text/html");

        //The term to search in resume column
        String term = req.getParameter("term");
        if (term == null)
            term="security";

        out.print("<html>");
        out.print("<body>");
        out.print("<H1>Search Result</H1>");
        out.print("<table border=1 bgcolor=lightblue>");
        out.print("<tr><th>ID#</th><th>Name</th></tr>");
        out.print("<tr>");
        try{
            Statement st = cn.createStatement();

            //search the term in resume column using SQL
            String query =
                "Select empno,ename from sam_emp" +
                " where contains(resume,'" +term+"')>0";

            ResultSet rs = st.executeQuery(query);

            while (rs.next()){
                out.print("<td>"+ rs.getInt(1)+"</td>");
                out.print("<td>" +
                    "<A HREF=http://sshah:8080/" +
                    "servlet/MyServlet?term=" +
                    rs.getString(1) +
                    " target=Document>" +
                    rs.getString(2) +
                    "</A></td>");
                out.print("</tr>");
            }

            out.print("</table>");
            out.print("</body>");
            out.print("</html>");
        }//try
        catch (SQLException se){se.printStackTrace();}
    }
}

```

Listing 4

```

package package1;

import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;
import java.io.*;
import oracle.jdbc.driver.*;
import oracle.sql.*; //for oracle.sql.BLOB
/**

```

This class reads the entire document from the résumé LOB column. It takes one parameter, term, to search a specific employee from the

sam_emp table and returns the document stored in that row.

```

* JDK 1.1.7, Oracle Thin JDBC 1.22 compliant driver
* with Oracle type extension classes (oracle.sql)
*
* @author Samir Shah
* @version 1.0
**/
public class MyServlet extends HttpServlet{
    Connection cn;

    public void doGet(HttpServletRequest req,
    HttpServletResponse res)
    {
        try{
            doPost(req,res);
        }catch (IOException ie){ie.printStackTrace();}
    }

    public void init(ServletConfig parml)
    throws ServletException
    {
        super.init( parml);
        try{
            DriverManager.registerDriver(
            (new oracle.jdbc.driver.OracleDriver()));
            cn =DriverManager.getConnection(
            "jdbc:oracle:thin:@sshah:1521:o8i",
            "scott", "tiger");
        }
        catch (SQLException se){se.printStackTrace();}
    }

    public void doPost(HttpServletRequest req,
    HttpServletResponse res) throws IOException
    {
        InputStream is=null;
        oracle.sql.BLOB blob=null;

        res.setContentType("application/msword");
        OutputStream os = res.getOutputStream();
        String term = req.getParameter("term");

        if (term==null)
            term="9001";

        try{
            Statement st = cn.createStatement();
            ResultSet rs = st.executeQuery
            ("Select resume from sam_emp"+
            " where empno='"+term");

            while (rs.next()){
                blob=((OracleResultSet)rs).getBLOB(1);
                is=blob.getBinaryStream();
            }

            int pos=0;
            int length=0;
            byte[] b = new byte[blob.getChunkSize()];

            while((length=is.read(b))!= -1){
                pos+=length;
                os.write(b);
            }
        }//try
        catch (SQLException se)
        {
            se.printStackTrace();
        }
        finally {
            is.close();
        }
    }
}

```



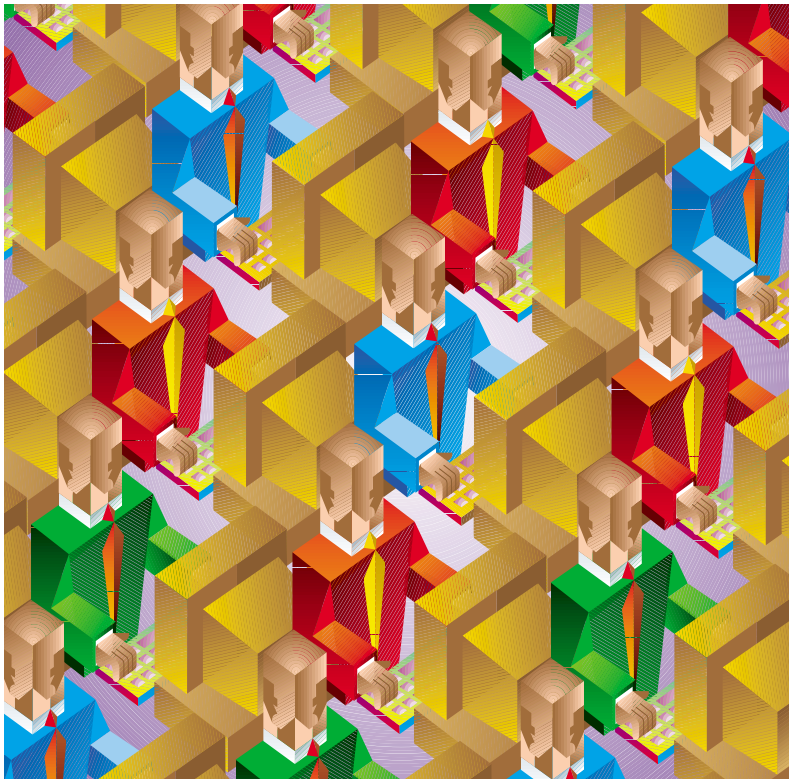
VSI

www.breezexml.com

Applying Patterns to JDBC Development

WRITTEN BY CHRISTIAN THILMANY

Developers at some point in their careers will find themselves standing at the whiteboard, trying their best to regurgitate some complex development design they've spent all night working on. This is usually done with a series of strange symbols, arrows and scribbles in an attempt to convey the clarity that may lie in the head of said developer (unless of course he or she doesn't know what exactly the design is supposed to look like). Either way, you have the same problem.



In an object-oriented fashion, how do you make tangible a design that's intangible yet very repeatable? How do you communicate – or in the latter case, how do you come up with – something you know in your heart you've done before but maybe in a slightly different way. And I'm not talking about UML. UML helps, but it provides only the hieroglyphics you may need to communicate your design. What you need is a repeatable way to show your design efforts without reinventing the proverbial design wheel by going through every step you originally used to solve the problem. If you're a developer or architect, for example, and you haven't gone through this, you probably will. Welcome to the world of patterns!

Rarely does a developer come up with the right design on the first attempt. Typically, each design must go through a bit of morphing before you can call it a sound one. And what about reusability? Isn't that one of the primary goals of object-oriented design? For both new and experienced designers the challenge of object-oriented design is difficult enough. In addition to being an outstanding communications tool, patterns help make the process of coming up with an elegant object-oriented design easier. More important, they help make a design reusable. Reusability applies not only to the objects themselves but also to the process used to come up with them. That's where patterns fit in – they help make object-oriented designs adaptable, sophisticated and, most important, repeatable.

A REAL-WORLD DATABASE SCENARIO – APPLYING BOTH DESIGN AND IMPLEMENTATION PATTERNS

Cruel World

www.cruelworld.com

When referring to patterns in this article I leave out the word *design* as I'm covering implementation patterns as well. Unlike what you find in some pattern books (although most I've read are excellent), I wanted to create a practical piece of code that uses both patterned design and patterned implementation – not another “gutted” bank application but an actual piece of code that could be used in a real-world business application. Something I might use myself with some adjustments. Now that's not to say the code I included for download is production ready (see the [JDJ Web site, www.JavaDevelopersJournal.com](http://www.JavaDevelopersJournal.com)); in fact, it isn't! However, I hope the included code will provide you with a base upon which to build something useful.

Initially I'll cover some of the most widely used patterns to implement a JDBC database connection “pooler,” something usually found in a distributed EJB development tool such as WebLogic or in other *n*-tier environments, such as Microsoft's Transaction Server. Even the newest flavor of ODBC has a connection-pooling mechanism built in. Nonetheless, using such a sophisticated environment for something like object pooling may be overkill; for pure Java environments, relying on ODBC connection pooling by using JDBC-ODBC as a solution isn't much better. Either way, I hope my connection pooler will provide some good examples of a few of the primary patterns used for database development *and* help you get your feet wet in the world of using patterns.

If you're new to patterns, one of the better places to start with is the “Gang of Four” (GoF). No, I'm not talking Spanky, Alfalfa and the gang, but Gamma, Helm, Johnson and Vlissides. Besides Christopher Alexander, who first spoke of design patterns when referring to buildings architecture, the Gang of Four, authors of *Design Patterns: Elements of Reusable Object-Oriented Software* (Addison Wesley), present some of the original thoughts on object-oriented design patterns from a language-agnostic viewpoint. Although they often mention C++ and Smalltalk, the book is geared to the patterns themselves, not the language used to write them. This is what makes patterns useful – their general applicability to all languages. For the Java developer this is all fine and good, but if you're like me, a little sample code always helps me to swallow dry subjects. That's why I highly recommend reading *Patterns in Java, Volumes I and II* by Mark Grand. These books cover patterns specifically articulated by the author and some of the more popular patterns covered in the Gang of Four books. They also cover other pattern originators such as Craig Larman whose “General Responsibility Assignment Software Patterns” (GRASP) present the more fundamental object-oriented ideas in the form of design patterns. Most important, Grand uses Java to exemplify each pattern. The source code is actually useful too.

What makes up a pattern? Or a design pattern in particular? Well, the Gang of Four breaks a pattern's description into textual sections that help explain the pattern in detail and show its context. They go on to explain that each pattern should highlight intent, motivation, applicability, structure (e.g., UML notation) and consequences, all of which help to describe the pattern as a whole. In addition, areas such as the pattern's design participants, its collaborations with other elements and its implementation help provide elemental detail so the pattern can be applied to a specific design. A pattern's sample code, known uses, alternative names and related patterns also contribute to its understanding.

Most of these areas are self-explanatory; for some, patterns may even overlap in meaning. Thus the gang also recommends that patterns be placed into certain pattern “classifications” based on principles such as the pattern's purpose and scope. These categories include creational, structural and behavioral. Each classification helps the developer look at a pattern in a particular perspective, which allows each pattern to fulfill different forms. The end result is that implementations are based not only on the pattern but also its classification. For example, some of the patterns I describe and apply in this article qualify as traditional GoF “creational” patterns and, when coupled with other complementary patterns, can be classified as distributed database patterns as well. I use the Singleton (GoF), Delegation (Grand), Observer (GoF) and Object Pooling

(Grand) in this way in the connection pooler code outlined below. I also refer to other related patterns such as Bounded Buffer (implementation), Exception Chaining (implementation) and Guarded Suspension (design) throughout.

I've limited the description of the patterns I use in the code example to a few “consolidated descriptives” that should provide enough information for you to grasp their meaning without my going into pages of detail. I'll show you how to apply these patterns to a database connection pooler component that can run either locally in a more traditional fat-client model or, in an attempt to create a more thin-client model, remotely, by using RMI to run the connection pooler on the server. The choice is yours. Although this example is written and tested in both environments, keep in mind that it has been designed primarily to run in a threaded RMI server. If run locally, you should make minor modifications to improve performance (e.g., remove synchronization qualifiers).

Okay, let's talk about a few common patterns that can be applied to a typical database implementation and, in my case, the connection pooler.

Singleton Pattern (GoF Creational Pattern)

- **Intent:** To ensure that the class has only one instance and to provide a global point of access to it.
- **Motivation:** Sometimes only one instance of an object can exist. For example, although there are many database connections, there's only one database connection pooler to control them.
- **Context:** An instance operation must be defined that allows clients access to the singleton object through this operation only. Using this single-access point of control improves namespace issues by eliminating the need for global references. Limited resources are typically controlled using this pattern so items such as connections can be controlled, pooled and dispensed with.
- **Solution and structure:** The structure is simple as it's implemented using a single class with one static instance operation that's used to return a unique instance variable. Typically, such classes have private constructors that don't allow external clients to dynamically instantiate this class, thus forcing them to use the instance operation only (see Figure 1).

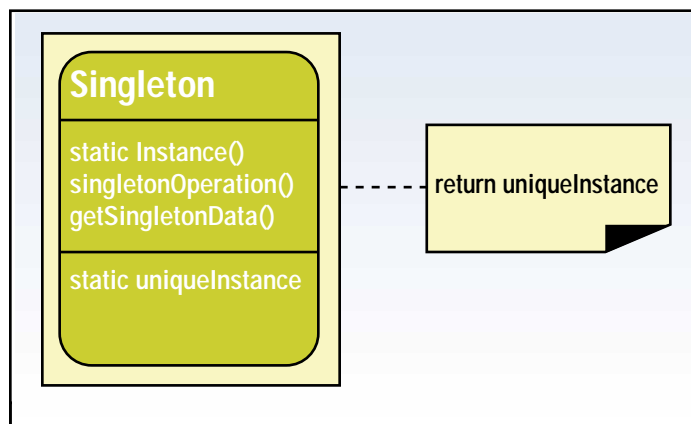


FIGURE 1 Singleton pattern UML diagram

- **Implementation:** Singleton implementations can vary as long as only one instance of the class is allowed and the creation sequence is accessible by all clients. In the database connection pooler, the pooler can run locally or remotely. Whether you run this code remotely using RMI or CORBA or just locally, the code should be equally effective at pooling connections. If run locally (without RMI), the connection pooler object follows the singleton pattern in a purist fashion since each database client can have only one instance of the pooler. This is controlled by using a static instance operation that returns the instance variable for the pooler that's allocated during first activation.

IAM Consulting

www.iamx.com

```

public static class ConnectionPooler implements . . .
{
    private static ConnectionPooler thePooler = new ConnectionPooler();
    private ConnectionPooler() { . . . }

    public synchronized static ConnectionPooler getDbConnectionPooler()
    {
        return thePooler;
    }
    . . .
}

```

When run remotely, which we'll cover later, the RMI server controls the instantiation of the connection pooler in a singleton fashion without requiring you to control single instantiation at the connection pooler level. In other words, although the connection pooler is treated as a singleton, it's done so through the RMI server, not the connection pooler itself; otherwise, each client would receive its own connection pooler when run remotely, which I'm trying to avoid.

The following shows how the singleton pattern can be implemented in various ways as long as the foregoing principles hold true.

```

public class AppServer extends UnicastRemoteObject implements . . .
{
    public static RemoteConnectionPoolerIF rcp = null;
    public synchronized SessionIF createSession(String dbURL) . . .
    {
        if (rcp == null) // only allocate one pooler per appserver
        {
            . . .
            rcp = (RemoteConnectionPoolerIF) DbConnection.ConnectionPooler.getDbConnectionPooler();
        }
        else
            . . .
    }
}

```

Object Pool Pattern (Grand Creational Pattern)

- **Intent:** To control the use of a limited resource, typically expensive to create, and/or a resource limited by the number of objects allowed to be created.
- **Motivation:** To increase application performance and prevent valuable and expensive resources from being exhausted, you must control them through pooling.
- **Context:** Aside from network connection initialization, creating connections to a database can be the single most expensive operation in an application. Pooling these connections provides dramatic application performance increases through repetitive database access. Controlling them also provides a means for keeping allocated resources under control (e.g., memory) in distributed architectures for the client as well as the server.
- **Solution and structure:** The structure of the pooling pattern contains the reusable objects, the client that uses them and a reusable object pool (see Figure 2). New instances are to be controlled and are usually limited so clients can reuse objects instead of creating new ones. For database connections it's usually necessary to assume that each reusable object is identical so that when run in a remote environment, with many different clients, each connection can be treated the same (this will lead to lengthy and usually heated security discussions with the database folks). They're all identical in nature. When run locally, it's possible to uniquely identify

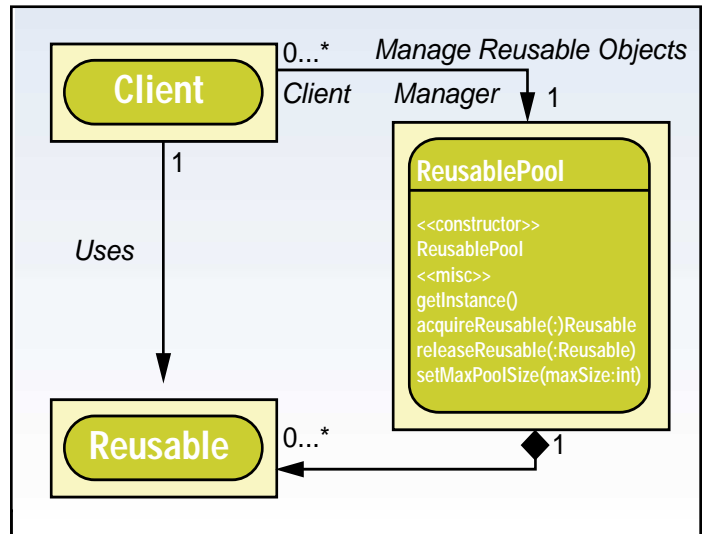


FIGURE 2 Object pool pattern UML diagram

each resource object that's managed, but the client is then required to pass in specific qualifiers (user ID and password), cluttering up the code.

- **Implementation:** In the connection pooler example, an RMI client (AppClientTest.java) represents a client. During session creation a client is given access to a session that allows it to acquire and release connections at will. The connection object is represented by the DbConnection class, which acts as a wrapper between the client and the actual JDBC code (see "Delegation Pattern" section below). The pooler, represented by a nested class within DbConnection, manages the connections by first initializing and caching them (caching is optional). Operations within the connection pooler allow each session to acquire and later release each connection. Whether new connections are continuously created on request or become "guarded" and wait for connections to be returned by other clients is an option controlled by the developer.

A NOTE ON GUARDED SUSPENSION

In the example a Bounded Buffer that first initializes the connection is created at startup, then controls the actual connections through "guarded suspension." The server will wait until connections exist in the pool before giving them out and, conversely, won't place connections into an already full connection pool. This allows the developer to better manage valuable resources on the server if desired. Guarded suspension can be considered either a design or implementation pattern, depending on the code. For this example, simply using the thread wait() operation along with method synchronization allows the connection pooler to control connections as mentioned.

```

if (usePreCachedConnections)
{
    if (usedSlots_ == 0)
    {
        long waitTime = MAX_WAIT_TIME;
        . . .
        for (;;)
        {
            try
            {
                wait(waitTime);
            }
            . . .
        }
    }
}

```

(Guarded suspension, a pattern in its own right, isn't considered part of the object pool pattern but is included here for demonstration purposes.)

Unify Corporation

www.ewavecommerce.com

Once received at the client, the client uses the connection like any other JDBC connection by issuing queries, updates or any other supported database operation. Each database operation is handled in a delegated fashion by JDBC, limited only by the JDBC driver used by the connection object. When a data operation is complete, the client releases the connection back to the connection pooler (releaseDbConnection()), which returns it to the pool at that time. This immediate acquire-and-release allows the client code to concentrate on the operation at hand, not the specifics of holding onto connections for performance reasons.

Precaching connections is optional and in our source code occurs during initialization, which is kicked off when the first client creates a session at startup. At that point a predetermined number of connections is acquired and placed into the pool for immediate availability. This allows access times to increase dramatically for additional online clients that require connections, thus boosting performance of the application as a whole. Guarded suspension is used when using precached connections only (see sidebar).

Delegation Pattern (Grand Fundamental Pattern)

- **Intent:** Delegation allows the developer to extend and reuse functionality of the existing class without using inheritance.
- **Motivation:** Avoiding inheritance may be optional or required, depending on the implementation. If a class can't be inherited or doesn't fit the "is a kind of" relationship to its parent, delegation should be used.
- **Context:** To pass a JDBC result set across the wire using RMI, it must either be "remoteable" (in the sample code RemoteResultSet extends RMI's UnicastRemoteObject and implements RemoteResultSetIF) and/or "serializable." This is required for RMI to allow a result set to be marshaled across the network. A JDBC result set, by default, doesn't fulfill either of these requirements, so delegation is used to assign remoteable calls across the network to its JDBC equivalent on the server. (See DbConnection.java - executeQuery() on the **DDJ** Web site.) For iterative next() queries (see AppClientTest.java on the **DDJ** Web site), this is *not* the most efficient way to query data since each record has to perform a network round-trip to complete. For a production environment it's recommended that a form of prefetching, similar to JDBC's own prefetching, is used to prequery results before passing them across the network. In fact, in most distributed database applications business services acting as connection clients typically fill that role from the server side, *not* the client side.
- **Solution and structure:** Using delegation, an object can assign operations to different objects at different times. Its structure (see Figure 3) is simply shown with the objects it delegates to as arrows leading to the delegator instead of to the traditional inheritance class diagram.

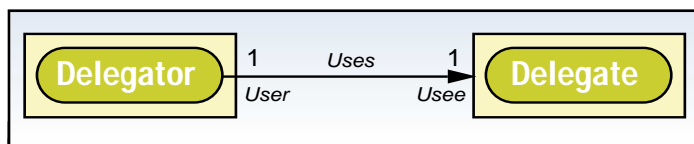


FIGURE 3 Delegation pattern UML diagram

- **Implementation:** For the connection pooler component to run on the server, a number of objects must be made remoteable (primitive remote without modification). These include the reusable connection object, connection pooler and result set that a query returns. Run locally, the connection pooler can use the default JDBC result set, but when run remotely I had to create my own version. To avoid having to reimplement a full result set from scratch, delegation can be used to assign remote requests from the delegator (RemoteResultSet) to the JDBC result set. When executeQuery() is called on the connection object, a new RemoteResultSet is created, passing into its constructor the result set that returned from JDBC's executeQuery(). Once the RemoteResultSet has the original JDBC result set, it simply delegates all calls to it (see previous paragraph on iterative next() calls).

```

public synchronized RemoteResultSetIF
executeQuery(String query) . . .
{
    stmt = connection.createStatement();
    rs = stmt.executeQuery(query);

    rsRemote = new RemoteResultSet(rs);

    return (RemoteResultSetIF) rsRemote;
}

```

Observer Pattern in Detail (GoF Pattern)

- **Intent:** To provide a mechanism that allows objects to dynamically subscribe to state change notifications from another object.
- **Motivation:** When client objects communicate with a server object of some kind, it would be beneficial for them to be made aware of server-side state changes in that object. This allows subscriber clients to react to the state changes in a more dynamic fashion, thus adding robustness to the application.
- **Context:** Suppose a client communicates regularly with a server object but that communication is time-critical. As with the connection pooler component, each initial connection to the connection pooler causes it to start up and fill the pool with connections. This could take several seconds, depending on how many connections the capacity is set for. Before this occurs it may be beneficial for said client to be notified when a server is online or going offline, thus giving it the opportunity to determine the time it will take to make such a connection. Not all clients may want this service, so the subscriber - or "observer" in this case - should have the option of observing only. The server, or observable object in this case, just notifies all observers of the state change, not caring which objects they're actually observing. Take heed, however: it could be a significant performance hit if hundreds of clients need to be notified. In this case the notification mechanism would be better served by running from a separate multicasting thread.
- **Solution and structure:** The Observer pattern is made up of an observer class that implements an interface containing a method the observable will call during notification. On the other side is the observable class, which implements an interface that the observer calls to register with the observable. The observer calls registerObserver() or addObserver() whenever that client wishes to receive notifications. The observable then keeps a list of observers and, during state changes, calls the notification method on the interface passed to it by the observer during registration. To stop receiving notifications,

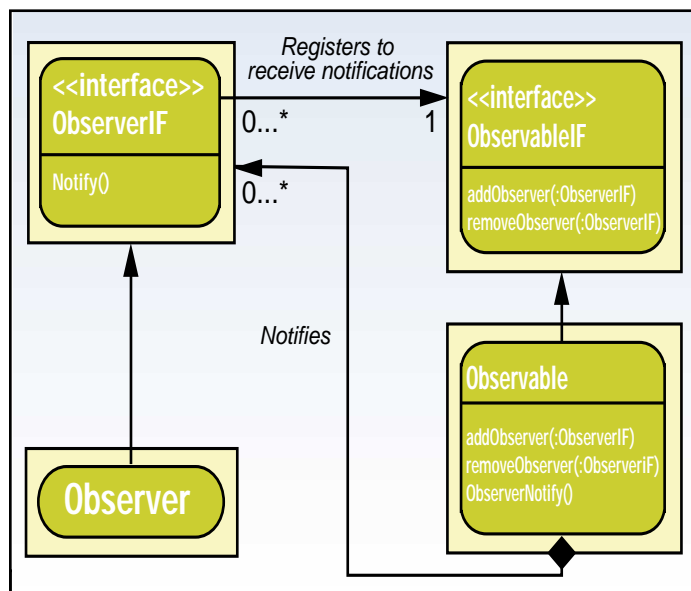


FIGURE 4 Observer pattern UML diagram

4th Pass

www.4thpass.com

the observer simply unregisters with the observable object in similar fashion (unregisterObserver() or removeObserver()).

- **Implementation:** The connection pooler implementation of this pattern is simple (see Figure 4). The AppClientTest implements RemoteObserverIF, which contains the method that the observable will call during a state change. In the RMI test client (AppClientTest), after the RMI server is started the client registers with the server by calling addObserver().

```
appServer = (RemoteAppServerIF) Naming.lookup(base_url + "/AppServer");
if (appServer == null)
{
return;
}

appServer.addObserver(this);
```

At this point the RMI server keeps track of this client. When a client calls closeSession() to close its own session, the server checks to see if there are any remaining connected clients. If there are still active clients, a message notification is sent to all observers that at least one session is active and the server won't shut down (until the last session closes). For this example I display a message only, but with a bit more work this could actually do something useful.

```
if (sessionCount == 0) // no more clients so close down the pool
    closeConnectionPooler(dbURL);
else
    remoteNotify("Active sessions still remain pooler still running...");
```

Other Database-Friendly Patterns Worth Mentioning

- **Guarded suspension (Lea):** This is implemented in the connection pooler code (see sidebar).
- **Balking (Lea):** Instead of using guarded suspension, the balking pattern could throw an exception instead of waiting for the correct state to occur.
- **Producer-consumer (Grand):** Use for asynchronous transactions in which a producer can push data objects onto a queue and a consumer can pop those objects for later processing (e.g., database error logger).
- **State (GoF):** Enhances the Session class in the connection pooler component to use a separate concrete state object, subclassed from an abstract state object. State-based behavior can be better decoupled using just one class to enhance reusability.

Developing a Database Connection Pooler

Now that I've covered the patterns from a structural perspective, I'll apply them to a fully implemented database connection pooler set of components (see Figure 5). This example is written as an RMI-based client/server Java application with the AppClientTest (RMI client) and the AppServer (RMI server) as the two major drivers of both client and server. This was written with Symantec's VisualCafé Enterprise Version 3.1 using JDK 1.2.2. I tested two Type 4 JDBC drivers: Oracle's Type 4 Thin Driver Version 8.16 against an Oracle 7 Database and the MS SQL Server 4 JDBC/Kona Type 4 Driver that works with the latest version of BEA's WebLogic going against MS SQL Server 7.0. For the example I connected to the "pubs" database and queried the authors' table with a reusable DbConnection object for testing.

Note: As mentioned earlier, executing queries remotely should be optimized to precache data that hasn't been implemented for this example. Precaching would avoid unnecessary network round-trips during iterative record traversals (e.g., using ResultSet.next()).

First unzip objectpool.zip (keeping directory paths is recommended) and edit both RunAppServer.bat and RunClient.bat to use your current class path (this must include the class path of the JDBC drivers you

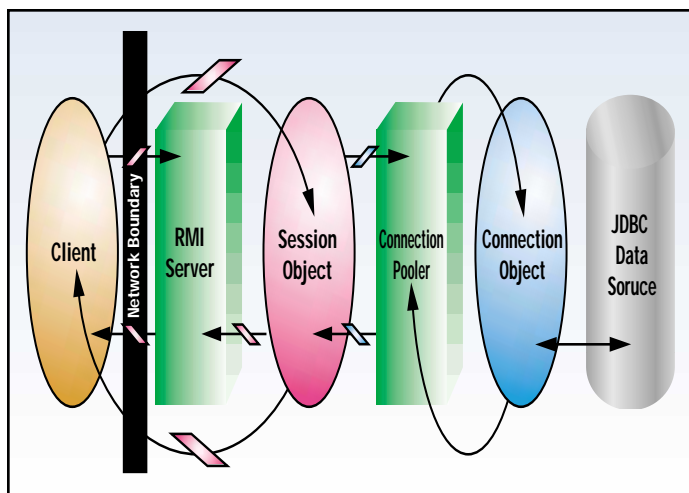


FIGURE 5 Connection pooler and related components – general architecture

decide to use). Once your JDBC driver is installed and tested, open the objectpool.vcp project (if you're using Café) and perform a full build; otherwise, compile individually.

If you're running from Café, turn off the automatic RMI compilation step. By default, VisualCafé will try and run "rmic" to build the RMI marshaling code during a full build – this may crash your system (this has been reported to Symantec/BEA support). To turn this off, from "project/options/compiler/compiler category/advanced," type "-normi" in the Custom Compiler Flags edit box and rebuild.

CLASS	FILE	DESCRIPTION
DbConnection	DbConnection.java	Main reusable connection object that wraps JDBC behavior. Used by client to perform all database operations, e.g., query, update, insert, et al.
ConnectionPooler	DbConnection.java	Nested inner class of DbConnection that implements the object pool, singleton, bounded buffer and guarded suspension patterns. Used to manage pool of DbConnection objects.
DbException	DbException.java	Exceptions class that supports exception chaining (implementation pattern).
AppClientTest	AppClientTest.java	Main RMI driver client used to mimic connection acquisition and release. Performs basic queries from command line.
AppServer	AppServer.java	RMI server used to drive ConnectionPooler and Session object creation. Initializes the ConnectionPooler to cache connections at start-up. Drives initialization and connection pooler cleanup. Implements the observer and singleton patterns.
RemoteResultSet	RemoteResultSet.java	Remote-friendly version of JDBC ResultSet that implements the delegation pattern.
Session	Session.java	Client-created class that holds client state. Used for acquiring and releasing connections remotely.

TABLE 1 Components of the connection pooler

MetaMata Inc

www.metamat.com

Go Online and Subscribe Today!

www.SYS-CON.com



or

Call 1-800-513-7111

Subscribe to the Finest Technical
Journals in the Industry!

GET YOUR OWN!

 SYS-CON
MEDIA

Once the project is built or all classes have been compiled, you can run RunRmic.bat from the command line to build the RMI marshaling code all at once. Now you're ready to run.

To run the app server from a command-line window, simply run RunAppServer.bat (edited with your current classpath). This should automatically start the RMI registry and the RMI server, which will begin "listening" for client connections. Once the server is running, from a separate command-line window, run RunClient.bat. This should begin the AppClientTest RMI client. When started, the AppClientTest will connect with the AppServer using RMI, create a session and begin acquiring connections using the session object. After the connections have been acquired, a simple query is made on the database, after which the user can select which action to perform next. When the first session is created, the AppServer will initialize the connection pooler and pass into it the number of connections to cache. This is where the database connections are first established and placed into the pool. This process may take a few seconds (for the first client), depending on how many connections you want to cache (see ConnectionPooler.Initialize() for details). Subsequent client connections (since they're now pooled) will be much faster.

You can fire up additional clients at any time using a separate command-line window. During the demo keep in mind that the connection pooler will block until some connections have been placed or returned into the connection pool, timing out if too many connections are requested at once (this can be adjusted). To the server each command-line window running AppClientTest represents a different client. This will demonstrate a simple concurrent environment. Note also that the connection pooler can be adjusted to run locally without RMI as well as with noncached connections. The choice is yours.

The connection pooler is actually made up of several components broken up into classes and files (see Table 1).

As mentioned earlier, the connection pooler is implemented as an inner class of the main DbConnection class using the singleton pattern. The AppServer, when running remotely, actually controls the singleton nature of the connection pooler by creating a new connection pooler only during the first connection. After that, each subsequent client request uses the existing connection held by an instance variable in the AppServer. When run locally, the connection pooler will run as a singleton as expected.

After initialization, the connection pooler class will use guarded suspension when connections have been cached (optional) to control the number of connections allocated. During operation, each client first requests a session object by calling createSession(), then a connection object (DbConnection) by calling acquireRemoteDbConnection() using the session object. Once a connection object is retrieved from the pool, the client can then use any of the remoteable operations on that connection (e.g., executeUpdate, executeQuery). When the client completes its operations, it will first release the connection by calling releaseRemoteDbConnection() using the session object, and finally close the session by calling closeSession(). The rest is up to you.

Summary

This article has focused on the real-world database use of design and implementation patterns. Although not quite ready for production, you can use the base code and apply it to most database applications, giving you yet another tool to avoid reinventing the design wheel and freezing at the whiteboard. ☺

AUTHOR BIO

Christian Thilmany is president of The eTier Group, Inc., in Houston, Texas. He has over 11 years' experience in distributed application architectures for Internet, intranet and client/server development using Java, C++, Visual Basic and more. Christian is a Microsoft Certified Solutions Developer.

christian@etier.com

JAVA DEVELOPERS JOURNAL.COM

Evergreen

www.evergreen.com/jdj.html

PropArgs — Every Programmer's Dream

This class is built for speed in application development, among other things

WRITTEN BY
GENE CALLAHAN
& ROB DODSON



The common factor in these questions is that they deal with the state of the program when it starts up. Answering the questions involves several problems that usually trouble programmers: saving and then restoring application state and user preferences; the normally painful process of command-line handling (Java doesn't have a standard class to do this); getting access to system properties, application properties and command-line arguments in a uniform manner; setting internal state of objects externally at runtime, that is, without recompilation; and accessing properties as various types (String, int, boolean, Vector, etc.).

Besides the benefits a reusable class for handling these issues brings to an individual program, our system staff saw a benefit in being able to centrally locate and manage companywide properties such as the network location of common servers, and the differences between our production and our test environments. In addition, we wanted to handle common application setup tasks such as parsing special command-line arguments (-?, -debug, -dump, etc.) and providing a standard formatted usage line for every program. By creating a class to handle all these problems, programmers can focus on solving the application problems at hand rather than the same old application start-up issues.

Solution: PropArgs Class

To solve these problems we developed a class called PropArgs (the name is short for "properties and arguments"). A property is a key = value pair of strings. String here means the Java String class at runtime and a simple text string of characters when stored on disk. In our implementation the characters in the key must be alphanumeric characters or periods. The value field

A common set of programming problems drove us to develop a Java class we call PropArgs. Consider the following questions a programmer may want answered about a program: Which RDBMS instance should data come from? Does this particular user have any personal preferences I should be setting? Should debugging code be executed during a particular run of a program? Are there different execution paths based on the current operating system? Should the programmer be operating in batch or interactive mode? What directory should disk output be written to?

may contain alphanumeric characters, spaces or punctuation. At runtime a property value may be looked up and its value converted into some other type. The conversion is done in the PropArgs class based on which of a set of get() calls the application makes. For example, the property size = 10 can have its value retrieved at runtime as a String, an int or a double, depending on whether the application calls get("size"), getInt("size") or getDouble("size"). Thus the application is shielded from having to know how the properties are stored inside PropArgs.

A group of properties is referred to as a property set. A set is named by a quintuple. Commas separate the elements of the quintuple. An element may be undefined, in which case a tilde is used as a placeholder. Each part of the quintuple has a domain. In order, the domains are Environment, Host, User, Application and Instance. ("Instance" allows multiple property sets per application for each user.) An example property set is test,~,~,OptionViewer. This designates properties for the test environment, on any host, for any user, for the OptionViewer application. Note that we don't require the Instance to be mentioned if it's undefined, as it is last and its absence doesn't present any difficulties in parsing the quintuple. Let's look at another example: prod,spica,rob,OptionViewer,1. This set is for the prod environment, on host spica, for user rob, for instance 1 of application OptionViewer.

Property sets are loaded at runtime in a particular order. This allows some properties to override others. As we load sets, starting with the most general and moving to the most specific, programmers are able to inherit common properties, then fine-tune this set for their application for specific users, and even for multiple instances per user, so that users can store multiple configurations. The load order is:

```
Env, host, ~, ~
Env, ~, ~, user, ~
Env, ~, ~, ~, app,
Env, host, ~, ~, app,
Env, ~, ~, user, app,
Env, host, user, app,
Env, ~, ~, ~, app,
Instance
Env, host, ~, ~, app,
Instance
Env, ~, ~, user, app, Instance
Env, host, user, app, Instance
```

If a particular set isn't found, it's just skipped, and the next in line is loaded. The special instance of save is used to store properties saved at runtime.

Property overriding is accomplished by simply overwriting older properties with newer values if the keys are the same. For example, if Speed = 19200 is defined in prod,~,~,modem and Speed = 56000 is defined in prod,~,rob,modem, then, if rob is running, the modem application Speed will be 56000, but for all others it will be 19200.

Features

In this section we'll discuss some of the main features of the PropArgs class. It has a rich set of methods for the application programmer.

COMMAND-LINE OVERRIDE

Properties specified on the command line override those loaded from the property database. This is a powerful idea. You can write your classes to have attributes and behaviors controlled from the property database. Nevertheless, runtime behavior of these classes can be modified simply by overriding the property on the command line. No recompilation is needed. We find ourselves putting more and more class configuration into properties for this reason. Also, because all properties are stored in one place, it's easier to adminis-

PointBase

www.pointbase.com/jdj

ter the company's applications and to control their behavior in a standard manner. An example of command-line overriding: if the property set `test,~,~,modem` contained the property `device = /dev/ser1`, to override it at runtime on the command line, you might say:

```
modem -device /dev/ser2
```

That's it! Note that the equal sign is not used on the command line.

STORAGE

Properties are currently stored in text files, with one property set stored per file. Only one spot in the PropArgs code knows about this location. They could just as easily be stored in a database and retrieved via SQL, or on a remote server and retrieved via a TCP/IP socket. Text files were chosen for ease of implementation and so as not to force all PropArgs clients to connect to a database. Text files also have the benefit of allowing processing by other text-handling tools. (We also provide a GUI for most common property editing tasks – the GUI is discussed in greater detail below.)

HANDLE DIFFERENT OPERATING SYSTEMS TRANSPARENTLY

There's only one operating system-dependent variable in all our numerous apps and that is in PropArgs. Programmers can specify

properties that are different depending on the OS the application is run on. This is done with property keys that begin with an OS name. Ideally you have a key = value pair for each OS you support, for example:

```
unix.database = /usr/data
win.database = c:\data
```

At runtime PropArgs determines which OS it's running on and does the following: all key = value pairs that start with the current OS name are kept, but with the OS name stripped off. All key = value pairs that start with other OS names are discarded. What's left is simply:

```
database = /usr/data
```

on UNIX, or, on Windows:

```
database = c:\data
```

VARIABLE \$VAR\$ SUBSTITUTION

String variable substitution is carried out at PropArgs constructor time. This is a handy way to share values and to save some typing. In the value part of any property you may refer to any previously defined key. Thus:

```
Foo = bar
Name = $Foo$
```

will set the value of Name to bar. You can see from this that order of property definition is important. PropArgs maintains the ordering of key = value pairs across saves and reloads.

STATIC INTERFACE

PropArgs provides a static interface so that classes that aren't directly passed a PropArgs instance at runtime are still able to get access to the application's properties. A static class variable called PropArgs.StaticPropArgs always points to the instance set in the method PropArgs.initStatic(), which is usually called in an application's main(). Any class may then access properties in the following manner:

```
PropArgs.StaticPropArgs.get("prop_key");
```

CHECK() METHOD

The PropArgs.check(exit) method makes sure that all the properties set in addUsage() calls are present. Currently it checks that a property is present and, optionally, that it has a value. Recall that command-line arguments need not have a value (e.g., -debug). If any arguments are missing or don't have values when they should, a usage line is printed to stderr, and if the exit flag is set, the application exits.

Check() also does a few other helpful things for the programmer. If there is a -debug property present, it turns on the printing of debugging information in our logging class. If there is a -? or -help command-line argument, a usage line is printed. Also, if there is a log.filename property present, it is passed to the logging class.

There is also a method usage() that can be called directly at any time to print the usage line. The addUsage() method is invoked as follows:

```
Props.addUsage("-file", false, "<filename>");
```

The second argument denotes whether the property is required. The last argument can be null if the property doesn't have a value but is just a boolean flag.

PUT()

The put(String key, String value), together with get(), are the workhorse methods of PropArgs. Put saves properties in an internal hashtable. Some internal debugging and housekeeping activities are done at put() time: saving the source of the property (file, application, internally created) for later debugging, and saving the order of the property (properties must be saved in the correct order). Dollar sign substitution as well as OS stripping is also done during put(), and all include processing.

GET()

The get(String key) is the main method used to retrieve a property's value. The most commonly used version of get returns the value as a String but there are methods to get the value as an int, long, double, boolean, Vec-

Generic Logic, Inc.

www.genlogic.com

Prosyst

www.prosyst.com

tor or another PropArgs instance. Most forms of get take a default value, which is returned if the key isn't found. For example,

```
String boss =
props.get("boss", "Rob");
```

would return "Rob" if no other boss was defined.

INCLUDE

PropArgs supports the notion of recursively including other property definitions, much like C or C++. Any key that begins with includeprops or includefile will cause PropArgs to recursively process that property set or file. Since in any single application all keys must be unique, to include more than one file or property set you must append some unique identifiers. Example:

```
includeprops1 = deve,~,~,ports
includeprops2 = deve,~,~,servername
includefile1 = /usr/tmp/externalstuff
includefile2 = /usr/tmp/morestuff
```

Including allows the sharing of properties among many applications. It also makes it easy to change the behavior of multiple applications in one place. For instance, if the name of a common server is changed, you need to edit it only once to update all of your applications.

SAVE

An important feature of the PropArgs class is the ability of an application to save properties that have been changed at runtime to the properties database. The most common use of the feature is for an application to save user prefer-

ences. For example, in some applications the user may be able to set color preferences, window positions, JTable column positions and so on. The user wants to use the same setting every time the application runs. The settings can be saved and then retrieved automatically by PropArgs the next time the program is run.

The saved properties are loaded last (see the load order above), thus overriding any default properties.

GETVECTOR()

A Java Vector can be instantiated from a property by PropArgs if it is of the form:

```
key = a,b,c,d
```

Calling:

```
Vector vec = PropArgs.getVector(key);
```

will return a Vector containing the elements "a," "b," "c" and "d."

GETPROPARGS()

A new instance of a PropArgs can be created from a key = value property pair if it looks like:

```
Info = a = b, 1 = 2, x = z, 3 = 4
```

Calling:

```
PropArgs newprops = props.get-
PropArgs("Info");
```

will do the job, and then:

```
newprops.get("a");
```

will return "b."

Creating a new PropArgs from a single property is a good way to use properties to set and save object states. A class could take a PropArgs instance as a parameter to its constructor and initialize itself from the key = value pairs. You may want to control exactly what properties a class sees, so rather than send it the main application instance of PropArgs, you pass one just for that class. A class can also save its state in a PropArgs instance, which can be turned into a single property in the main application's complete property set. This may seem convoluted, but we use it frequently and it is quite powerful.

PROPARGSOBJFACTORY

To save and load more complicated classes to and from properties, we designed a separate class called PropArgsObjFactory. To use this class you simply need to write put() and get()

methods to convert your class to and from string form. For example, to store and load the Java Color class as a property, the interface might look like:

```
Color getColor(String key, PropArgs
props)
Void putColor(String key, Color
color, PropArgs props)
```

So a call like:

```
PropArgsObjFactory.putCol-
or("Button.color", color1, props);
```

might produce a key = value pair like:

```
Button.color = 255,100,100
```

The get method just needs to parse the RGB string and instantiate a Color.

PROPARGSGUI

We've written a Java-based GUI to allow developers and system administrators to edit properties. This has made controlling application parameters easy. It's also become an excellent debugging tool. For example, with the GUI, if a user complains of a problem, we can see just what the setup for the application was and possibly fix it directly in the GUI.

Listing 1 is an example of the typical PropArgs setup in main().

Typical application usage might look like Listing 2.

Summary

The PropArgs class has benefited us in several ways:

- It's sped up application development by making it easy to access all the various properties in the Java runtime environment.
- It's taken away the drudgery of parsing and checking command-line arguments.
- It's made it easy to externally control the internal behavior of applications and classes.
- By centrally locating all the application properties for the entire company, it's easy to make changes that affect all or many applications at once.

If you're interested in using the PropArgs class for your project, the source code is available at www.robodson.net/java.

• • •

Thanks to the other developers who contributed to the design of PropArgs, especially Marlon Guarino.

gcallah@erols.com robodson@erols.com

AUTHOR BIOS

Gene Callahan, president of St. George Technologies, designs and implements Internet projects. He has written articles for several national and international industry publications.

Rob Dodson is a software developer who writes options-trading software in Java and C++ for OTA Limited Partnership.

Previous projects include weather analysis software, tactical programs for Navy submarines and code for electronic shelf labels.

Listing 1

```
public static void main(String[] args)
{
    PropArgs props = new PropArgs("App-
Name",args,null);
    Props.initStatic(props);

    props.addUsage("server", true, "<host>",
"hostname");
    props.addUsage("file", false, "<file name>");

    props.check(true);

    String server = props.get("server");
    String file = props.get("file","default_file");
}
```

Listing 2

```
Boolean debug = props.getBoolean("debug");
int size = props.get("size",20);
Vector names = props.getVector("userlist");
Color background =
PropArgsObjFactory.getColor(props.get("backcol-
or"));
props.put("size","100");
props.putDouble("pi",3.14159);
props.save();
```

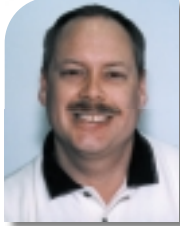

Sic Corporation

www.sic21.com

Building Enterprise Beans with VisualAge for Java

Tools that eliminate the frustration of developing EJBs by hand

WRITTEN BY
LUCY S. BARNHILL,
ANGUS MCINTYRE &
ROB STEVENSON



Enterprise JavaBeans (aka EJBs) are fast becoming a mainstay in Web-based business applications. They're not trivial to develop, though – at least not if you're developing them by hand and ignoring the EJB tools already available to automate (and thus simplify) your development tasks.

If you earn your living as a professional Java programmer, you know that enterprise beans are nonvisual, server-side software components that conform to Sun Microsystems' EJB specification. Enterprise beans allow you to develop platform-neutral, distributed applications that run on virtually any EJB-compliant server.

EJB technology simplifies server-side application development by delegating many of the common system-level programming tasks – transactional semantics, data persistence, security and workload management – to the EJB server providers. This allows you to focus primarily on the application business logic when developing enterprise beans.

Enterprise beans and their many benefits have been well documented in past issues of *Java Developer's Journal*. A complete description of EJB technology and its advantages is found in Sun Microsystems' EJB 1.0 and 1.1 specifications at <http://java.sun.com/products/ejb/docs10.html> and <http://java.sun.com/products/ejb/docs.html>.

While they play an increasingly important role in the development of enterprise Web applications, developing EJBs presents some significant challenges. After we examine some of the challenges, we'll look at a solution that can help simplify development. This solution is found in the WebSphere application development tools that are part of IBM's VisualAge for Java product, which we'll test-drive with a short, hands-on tutorial.

The Challenges

Despite the many advantages of enterprise beans, they are technologi-

cally complex and developing them is generally not a simple task. To create an enterprise bean, you must follow a set of interfaces defined by the EJB specification. For example, in addition to defining an enterprise bean class, you must define both home and remote interfaces for each enterprise bean. The latter defines the client's view of the enterprise bean's business methods; the former defines the client's view of the bean's object life-cycle. This involves such events as the creation and removal of the enterprise bean.

You also need to ensure that the methods defined in the enterprise bean's interfaces and classes are kept consistent. And if you're creating an entity bean, you must define its persistence fields and map them to a persistent datastore, such as a relational database. Once you've created the enterprise bean, you need to target it to a specific bean container by generating the implementation classes for the home and remote interfaces. Then you need to test the home and remote methods. Finally, once testing is complete, you need to package the bean for installation on a production server.

This isn't an exhaustive look at the challenges associated with developing enterprise beans, but it should give you a feel for their underlying complexity. Fortunately, there are ready-made solutions that can help you develop enterprise beans quickly and effectively.

One Solution

Although you can develop enterprise beans by hand, it's generally faster and easier to use a set of application development tools specifically designed for the task that mask much of the complexity. The IBM WebSphere tools built into VisualAge for Java enable you to use VisualAge interactively with other Web

development products, such as WebSphere Studio and the WebSphere Application Server. The tools include:

- WebSphere Test Environment
- JSP/Servlet Development Environment
- EJB Development Environment

The WebSphere Test Environment allows you to test servlets, JSP files and enterprise beans in a runtime environment that's essentially the same as that provided in the WebSphere Application Server. This enables you to develop code for deployment to the WebSphere Application Server or to other application servers from non-IBM vendors. The JSP/Servlet Development Environment lets you run, monitor and debug servlets and JSP files that you've created in WebSphere Studio or other Web development products. And the EJB Development Environment enables you to develop enterprise beans and associated EJB components. Since enterprise beans are the focus of this article, we'll take a closer look at this environment.

The EJB Development Environment

You can use this specialized environment to develop and test enterprise beans that conform to the EJB specification. In the VisualAge for Java Workbench, the EJB page is the heart of this environment. It's where all your enterprise beans and related components reside, and it's where you accomplish all your enterprise bean development activities. In the EJB page (shown in Figure 1) you can access and run the EJB Development Environment tools, as well as write and edit any required business logic.

The EJB Development Environment tools simplify your development tasks by generating most of the infrastructure code for your enterprise beans. Specifically, they:

- Create EJB groups to hold your enterprise beans.

This article, which discusses some of the implications of EJB development, includes a tutorial that allows hands-on sampling of some EJB development tools by installing the IBM VisualAge for Java CD found in next month's issue of JDJ.

Softwired

www.softwired-inc.com/ibus

- Create new session enterprise beans or entity (BMP or CMP) enterprise beans.
- Inherit properties from other enterprise beans (e.g., CMP fields, methods, control descriptor attributes).
- Import existing enterprise beans.
- Add home and remote interfaces.
- Build persistence into enterprise beans by adding, defining and mapping CMP fields.
- Create and map associations between CMP entity beans.
- Set deployment and control descriptors.
- Generate deployed classes.
- Create and edit access (adapter) beans.
- Verify that enterprise bean code is consistent and conforms to the EJB specification.
- Maintain source code and generated code using the built-in team and versioning capabilities.
- Test and debug enterprise beans using a generated test client and a test server.
- Export your code for deployment to production servers.

Although all elements of the EJB Development Environment play an important role, access beans and the test client warrant some special attention.

Access beans, sometimes known as *adapter* beans, serve as JavaBean wrappers for your enterprise beans. Generated by a wizard and typically used by client programs such as JSP files, servlets or even other enterprise beans, they allow you to hide the home and remote interfaces of an enterprise bean and adapt them to the JavaBeans programming model. This simplifies the interface between enterprise beans and servlets or JSP files by providing a Java-

Beans interface that's recognized by all Java developers. Access beans introduce advanced local caching of enterprise bean attributes, which reduces the number of remote calls and provides faster access to enterprise beans.

The test client is an application you can generate automatically and run to test each enterprise bean that's running in the EJB test server. It features its own user interface and allows you to test individual methods in the home and remote interfaces of an enterprise bean. It makes testing enterprise beans as easy as testing local Java programs and saves you the effort of coding your own test client.

More detailed information about the EJB Development Environment and other WebSphere tools is found in "WebSphere Support in VisualAge for Java 3.0" at www.software.ibm.com/vad.

The Tutorial

In this tutorial you'll create and test an enterprise bean using some of the core tools in the EJB Development Environment. For our purposes assume that a small, unnamed bank has asked you to develop a simple banking application where, for "tax purposes," information about customer bank accounts is limited to a bank account number and the balance in the account.

You decide to create an enterprise bean that enables customers to create their own bank accounts and specify the account numbers, query their bank balances, and make deposits and withdrawals. (Customers should also be able to delete their accounts in case they

suddenly need to take an extended trip out of the country!)

Since data in a bank account needs to persist after a customer banking session ends, you need to create an entity enterprise bean. You don't want to spend a lot of time writing the logic required to store the data, so you decide to use a container-managed persistence (CMP) entity bean, which delegates the data storage tasks to its container.

Since you're a Java whiz and familiar with EJB technology, you know that creating, finding and deleting accounts are handled by the home interface of the enterprise bean, which extends the `javax.ejb.EJBHome` interface defined by the EJB specification. You also know that depositing, withdrawing and obtaining a balance are handled by the remote interface of the enterprise bean, which extends the `javax.ejb.EJBObject` interface. (This interface contains the business methods that a client application can call on an enterprise bean. The client application has access only to the methods that a developer chooses to expose through the remote interface. It doesn't have direct access to the enterprise bean.)

Now you know the "history" behind the tutorial, you're ready to begin your short odyssey into enterprise bean development.

Step 1: Prepare for the tutorial.

To allow you a hands-on experience of the tutorial, next month's issue of *JDJ* includes the following two CDs:

- IBM VisualAge for Java, Entry Enterprise Edition for Windows
- IBM DB2, Universal Database Personal Edition for Windows

To continue with the tutorial, you need to install these products, create the sample database and perform some basic setup tasks. Installation and setup instructions for the tutorial can be found at www.ibm.com/software/vadd/homedata/va-db2.

Step 2: Create the project and package.

Now that you've finished preparing for the tutorial, you can create the project and Java package. (A project is a receptacle used to hold a collection of related Java packages, much like a directory in the file system.)

1. Open the VisualAge for Java Workbench, click the Projects tab, then click the Add New or Existing Package to Workspace icon.
2. In the Add Package SmartGuide Project field, type SimpleEJB.
3. In the Create a new package named field, type netbank.
4. Click Finish. The new project and package are added to the Workbench.

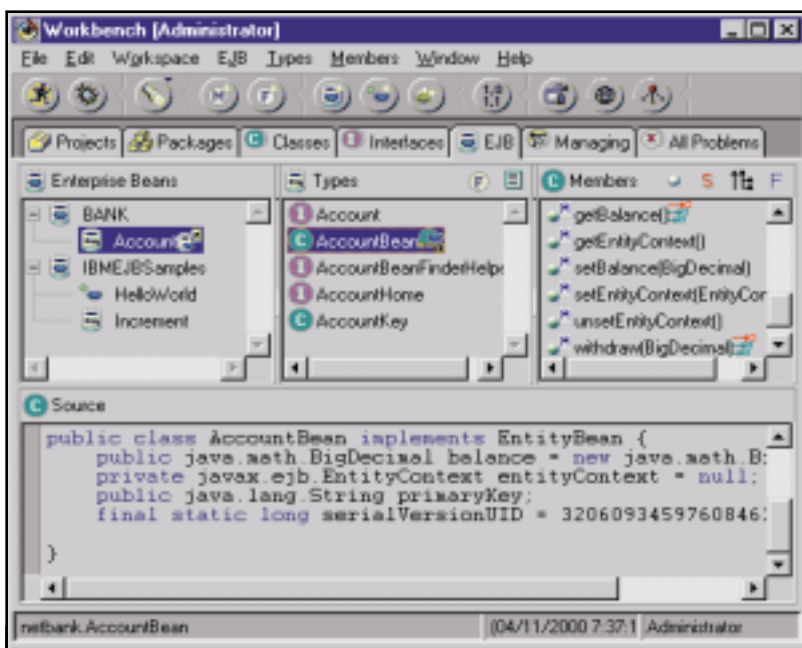


FIGURE 1 The EJB page

KI Group Inc

www.klgroup.com/greats

Step 3: Create an EJB group.

In this step you create an EJB group, which is simply a receptacle to hold and organize related enterprise beans.

1. Click the EJB tab to open the EJB page of the EJB Development Environment.
2. Click the Add EJB Group icon to open the Add EJB Group SmartGuide.
3. In the Project field, type SimpleEJB, and in the Create a new EJB group named field, type BANK.
4. Click Finish to generate the code into the associated project.

Step 4: Create the enterprise bean.

Now that the EJB group is created, you can create the enterprise bean. Since you need to store persistent data for the account balance and you don't want to write your own logic to store the data, you'll create a CMP entity bean.

1. Click the Add EJB Bean icon to open the Create Enterprise Bean SmartGuide.
2. In the Bean name field, type Account. (AccountBean appears as the default entry in the Class field.)
3. In the Bean type field, select Entity bean with container-managed persistence (CMP) fields.
4. Click Next. The Define Bean Class Attributes and Interfaces page appears. (Note that, by default, AccountHome is displayed in the Home interface field, Account is displayed in the Remote interface field and AccountKey is displayed in the Key class field.)

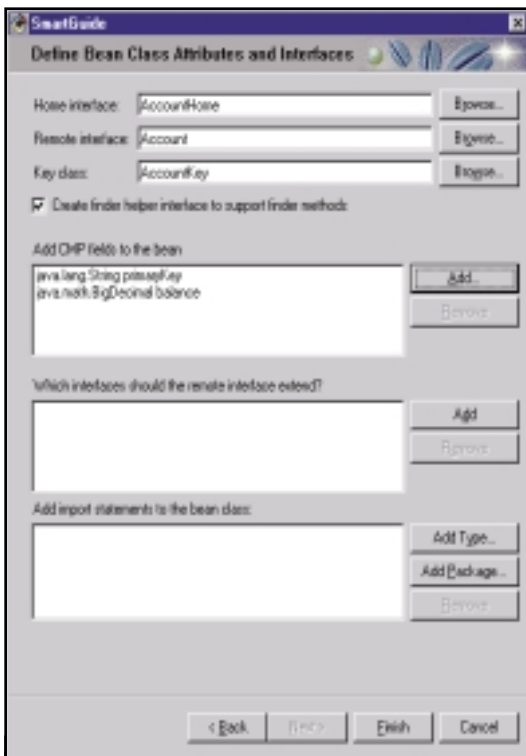


FIGURE 2 The Define Bean Class Attributes and Interfaces page

5. Ensure that the Create helper finder interface to support finder methods checkbox has been selected.
6. Click Add beside the Add CMP fields to the bean list box to open the Create CMP Field SmartGuide. You'll use this SmartGuide to set up the fields to store the bank account number and the balance in the account.
7. In the Field Name field, type primaryKey, then, in the Field Type field, select java.lang.String. This field will be used to find or create new instances of AccountBean.
8. Select the Key Field checkbox, then click Finish to add the CMP field and close the SmartGuide.
9. Open the Create CMP Field SmartGuide again so you can add another CMP field to store the balance in the bank account. In the Field Name field, type balance. Beside the Field Type field, click Browse to open the Field Type dialog box, then, in the Pattern field, type BigDecimal and click OK to close the dialog box.
10. In the Initial Value field, type new java.math.BigDecimal(0). Ensure that the Access with getter and setter methods checkbox is checked and that the public radio buttons are selected for the getter and setter methods. (In Java, field values are retrieved by a getter method and set with a setter method. The SmartGuide generates the code to both get and set the balance in the account.)
11. In the Create CMP Field SmartGuide, click Finish to add the CMP field. The Define Bean Class Attributes and Interfaces page should now look like Figure 2.
12. Click Finish to generate the enterprise bean.

The code in Listing 1 appears in the Source pane.

Step 5: Add the required methods.

Now you need to add some new methods so you can deposit and withdraw from the account.

1. In the Types pane of the EJB page, select AccountBean.
2. Click the Create Method or Constructor icon to open the Create Method SmartGuide.
3. Ensure that Create a new method is selected, then click Next to open the Attributes SmartGuide.
4. In the Method Name field, type deposit, then, beside the Return Type field, click Browse to open the Field Type dialog box.
5. In the Pattern field, type BigDecimal and click OK to close the dialog box.
6. Now you're ready to add and define a

- parameter for the amount of money to be deposited in the account. Click Add to open the Parameters dialog, then, in the Name field, type amount.
7. Select the Reference Types radio button, then, in the field below, type BigDecimal to specify BigDecimal as the return type.
8. Click Add, then click Close to close the Parameters dialog.
9. Select Finish to generate the code. The method will take the BigDecimal value amount as a parameter, then return an updated BigDecimal value balance.

The following code appears in the Source pane:

```
public java.math.BigDecimal
deposit(java.math.BigDecimal amount)
{
    return null;
}
```

Step 6: Implement the deposit and withdraw methods.

Next, you implement the deposit and withdraw methods to perform the business logic. The deposit method will retrieve the current balance using the getBalance method, add the input amount to the balance, then use the setBalance method to store the new balance.

1. In the Source pane replace the existing code with the following code:

```
public java.math.BigDecimal
deposit(java.math.BigDecimal amount)
{
    setBalance(getBalance().add(amount));
    return balance;
}
```

2. In the Source pane right-click and select Save.
3. In the Source pane add the withdraw method to the program by changing the word *deposit* to *withdraw*, then changing the word *add* to *subtract*.
4. In the Source pane right-click and select Save to create the withdraw method.

Step 7: Promote the methods to the remote interface.

Now that you've finished your Java coding, you need to promote the getBalance, deposit and withdraw methods to the remote interface so you can manage your account. (Note that you don't promote the setBalance method to the remote interface. This would enable customers to set their own bank balance for any amount they like!)

OOPSLA 2000

www.oopsla.acm.org

1. In the Members pane right-click the `getBalance` method and select `Add to > EJB Remote Interface`. An icon appears beside the method to indicate that it's now part of the remote interface.
2. Using the same procedure, add both the `withdraw(BigDecimal)` and `deposit(BigDecimal)` methods to the remote interface.

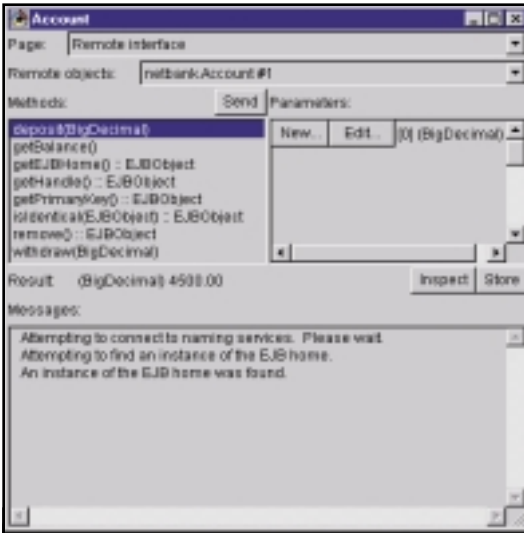


FIGURE 3 Remote Interface page of the test client

Step 8: *Generate the schema, map and database table.*

Since your EJB group now contains an enterprise bean, you can use the top-down approach to generate a schema and map from the EJB group. In this approach the enterprise bean design determines the database design. The generated schema contains one table for each CMP entity bean in the EJB group. In the table each column corresponds to a CMP field and the generated mapping maps the field to the column.

1. Open a Windows NT command window and issue a `db2start` command to ensure DB2 is running.
2. In the Enterprise Beans pane right-click the BANK group and select `Add > Schema and Map from EJB Group`. This creates the default schema.
3. Click the `Open Database Schemas` browser icon to open the Schema Browser. In the Schemas pane select BANK, then, in the Tables pane, select Account. This displays the columns in the table.
4. To export the schema, from the Schemas menu select `Import/Export Schema > Export Entire Schema to Database`. The Database Connection Info dialog box opens.
5. In the Connection Type field, select `COM.ibm.db2.jdbc.app.DB2Driver`.
6. In the Data source field, type `jdbc:db2:sample`, then click OK. The Console window opens to confirm

- that an Account table was created with a `primaryKey` column for the account number and a balance column for the balance in the account.
7. Close the Schema Browser.

Step 9: *Generate the deployed code and the test client.*

Now you need to generate the deployed code, which consists of the home interface code and the communications code (stubs and ties) that serves as the middleware used to connect the client to the server. You also need to generate the test client to test the enterprise bean.

1. In the Enterprise Beans pane right-click BANK, then select `Generate > Deployed Code` and wait for the code to be generated.
2. In the Enterprise Beans pane right-click BANK again and select `Generate > Test Client`. This automatically generates the test client code and a default user interface so you can test the enterprise bean without coding a full-blown user interface.

Step 10: *Publish the enterprise bean to the EJB test server.*

Now that you've generated a test client, you need to publish the enterprise bean to the EJB Server Configuration browser, set some properties, then start the required servers.

1. In the Enterprise Beans pane right-click BANK and select `Add To > Server Configuration`. The EJB Server Configuration browser appears.
2. In the Servers pane right-click Persistent Name Server and select `Start Server`.
3. In the Console window ensure that the Persistent Name Server process is selected in the All Programs pane and wait until the message "Server open for business" appears in the Output pane. (If the Persistent Name Server fails to start and you're using a disconnected laptop computer, install the loopback adapter from your Windows NT CD, configure it, then try starting the Persistent Name Server again.)
4. In the Servers pane of the EJB Server Configuration browser, right-click EJB

Server (server1) and select Properties to open the Properties dialog box.

5. In the Data Source field change the existing value from `jdbc:db2:sample-DB` to `jdbc:db2:sample`.
6. In the Connection Type field, ensure that `COM.ibm.db2.jdbc.app.DB2Driver` is selected, then click OK to close the dialog box.
7. Right-click EJB Server (server1) and select `Start Server`. In the Console window select the EJB Server process. Wait until the message "Server open for business" appears.

Step 11: *Run the test client.*

1. In the EJB Server Configuration browser expand the BANK group and select the Account enterprise bean, then click the Run Test Client icon. The Connect page of the test client appears.
2. On the Connect page click the Connect button. The Home interface page appears.
3. Ensure that `create(String)` is selected, then, in the Parameters field, type `Acct355` and click Send. This creates an instance of a bank account with `Acct355` as the primary key. The Remote interface page appears.
4. Ensure that the `deposit(BigDecimal)` method is selected, then click New to open the Constructors dialog.
5. Select the constructor `new BigDecimal(String)`, then, in the Parameters field, overwrite the null value with the value `4500.00`.
6. Press Send, then press Done to close the Constructors dialog.
7. In the Remote interface page press Send. In the Result display field a message confirms that the amount of `4500.00` has been deposited into the account, as shown in Figure 3.

Congratulations! You've just created and tested a fully functional enterprise bean that you can use in the simple banking application requested by that small, unnamed bank! 🎉

lucysb@us.ibm.com / rstevens@ca.ibm.com / mcintyre@ca.ibm.com

AUTHOR BIOS

Lucy S. Barnhill is a development manager at the IBM Research Triangle Park facility in Raleigh, North Carolina. She manages the Development and Information Development teams working on VisualAge persistence tools.

Angus McIntyre is the marketing manager for VisualAge for Java. He has 16 years of experience at the IBM Toronto Lab.

Rob Stevenson, an information developer at the IBM Toronto Lab, writes online help and publications for the VisualAge for Java product.

Listing 1

```
import java.rmi.RemoteException;
import java.security.Identity;
import java.util.Properties;
import javax.ejb.*;
/**
 * This is an Entity Bean class with CMP fields
 */
public class AccountBean implements EntityBean {
    public java.math.BigDecimal balance = new java.math.BigDecimal(0);
    private javax.ejb.EntityContext entityContext = null;
    public java.lang.String primaryKey;
    final static long serialVersionUID = 3206093459760846163L;
}
```


Starbase

www.starbase.com

Sterling Software

www.cooljoechallenge.com

Sterling Center Spread p/u

CLIENT SELF-CONTAINED APPLET USING SWING



Java Swing components allow you to execute applets inside the secured well-defined browser environment while creating an effective GUI outside the browser's frame

WRITTEN BY THOMAS CZERNIK & ROLF KAMP

Most Web-based applications today confine users to the frame of their browser, restricting them to viewing only one Web page at a time. Technologies such as JavaScript make additional browser windows possible, but this approach doesn't enable the kind of customized menus, toolbars and windowing features available in a traditional client/server application. Java Swing provides additional capabilities such as the ability to create windows outside the browser that have the look and feel of a traditional client application. The techniques described in this article can be used to migrate traditional client/server applications to a Web-based environment.

Event-monitoring applications that display and continuously update event information in real time stand to benefit from such an approach. For example, applications that monitor changes in financial data, network activity or weather conditions and require the continuous display of information must be updated in real time based on events occurring on the server. These applications usually require long-running applets that monopolize the browser, which in a Web-based environment oblig-

Fiorano

www.fiorano.com

es users to open a second instance of the browser if they want to view additional information. This creates a need for Web-based applications that behave similarly to typical client/server multiwindowing applications because these don't monopolize the browser window space.

A combination of Java Swing and CORBA provides the ability to build applications of this nature, referred to as "self-contained client applets." These applets use CORBA to communicate with a server in real time and Java Swing to create an effective GUI. Java-enabled Web browsers act as a common platform from which applets using CORBA may be launched. Once the self-contained client applet is launched, the user has the option of visiting other Web pages or minimizing the browser.

One integral part of this applet is the use of CORBA, as demonstrated in our November 1999 article "Real-Time Web-Based Applications with Java and CORBA" (*JDI*, Vol. 4, issue 11). An ORBlet-enabled Web browser permits the execution of CORBA-enabled applets.

The CORBA callback is a well-defined, easy-to-use technique for developing real-time, Web-based clients. Clients wanting to receive real-time data register with the server by passing it a client object reference. The server stores a reference to each client that receives data in real time. As the state of the server changes, client references in the server are used to send data to the client. Callbacks are an effective way to have clients receive data from the server without having to poll the server. Another approach that can be used to notify clients of events is the CORBA-defined Event Notification Service. Remember to keep performance requirements in mind when deciding which approach to use.

Swing components allow you to execute applets within the secured, well-defined browser environment while creating an effective GUI outside the browser's frame. Applets that create windows outside the browser's frame make it possible for users to view other Web pages or minimize the browser as the applet executes. Combining applets and CORBA allows an application to behave like a traditional client/server without requiring any software installation or configuration on the client. The user simply accesses a Web page containing a CORBA-enabled applet that binds to the CORBA server, establishing a client/server connection. Creating applications in this manner gives you the best of both worlds and allows you to create powerful, effective applications.

Overview of Swing

In an effort to make Java more consistent across platforms, the Java Foundation Classes were created jointly by Sun Microsystems, IBM, Netscape and Lighthouse Designs (now a part of Sun Microsystems). JFC is a suite of GUI classes that includes Abstract Windowing Toolkit (AWT), Drag and Drop, Java 2D, Accessibility and Swing. Based on AWT and written entirely in Java, Swing provides users with a GUI that looks consistent across platforms and provides developers with a GUI that performs identically across platforms. JFC is part of Java 2 and can be integrated with any JDK version newer than 1.1.5.

Java Swing components are based on Netscape's Internet Foundation Classes. Written in Java, the IFC was created to simplify the creation and management of Java GUI components and greatly extends the AWT. The AWT Container class was extended by Swing with the introduction of the JComponent, the superclass of most Swing components. This class is a member of the javax.swing package.

A "J" precedes all component classes in Swing - JLabel, JButton, JTree and so on. Classes derived from JComponent provide various characteristics including platform-independent presentation, shortcut keys (termed *mnemonics*) and common event-handling capabilities. Swing components derived from JComponent are written entirely in Java and their look and feel are independent of the platform they're executed on, which is why they're termed *lightweight* (see Figure 1).

A Java program that uses AWT has a different look and feel depending on the platform it's being executed on. Running on a Macintosh, it looks and feels different from the same program executing in a Microsoft Windows environment. These differences may cause a GUI component to vary drastically in appearance depending on which platform the component is used on - the amount of space a component occupies may dif-

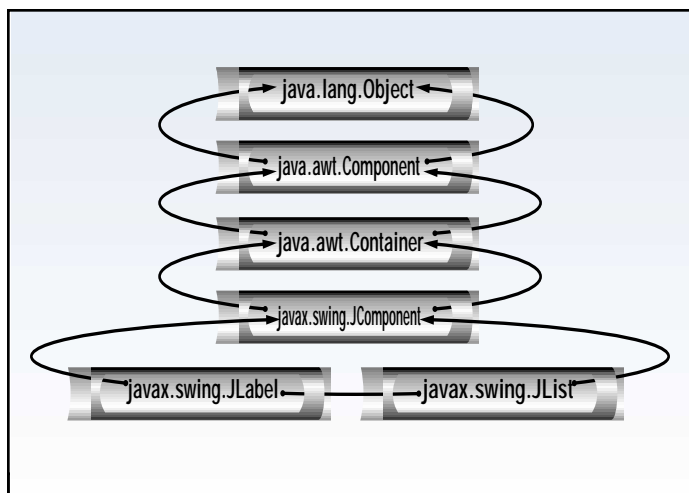


FIGURE 1 Inheritance diagram for Swing lightweight components

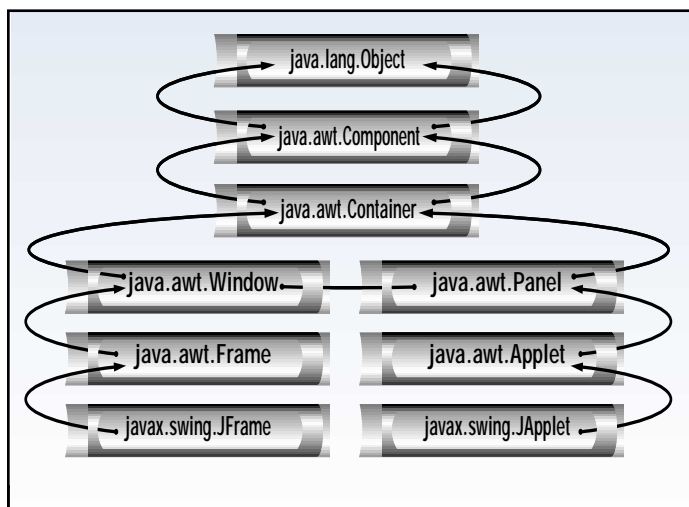


FIGURE 2 Inheritance diagram for Swing heavyweight components

fer, for example. Swing's creators took the best features of existing GUIs and implemented them in a look and feel termed *Metal* - Swing uses it by default - that delivers a uniform look and feel across platforms. There's also a *Windows* or a *Motif* look and feel that you can specify. The Swing class javax.swing.UIManager is used to manage the default look and feel of a Java Swing program.

Not all "J" classes are lightweight. The Swing containers JFrame, JDialog, JWindow and JApplet are heavyweight containers. These classes are derived not from JComponent but from an AWT class and they're termed *heavyweight* because their presentation is less flexible than lightweight components and is linked directly to the native platform's windowing system. These components are constrained (or weighed down) to the native platform's GUI capabilities. Heavyweight components don't necessarily perform or look the same on different platforms. The heavyweight component's AWT peer manages the interactions between the native platform and the Swing component (see Figure 2).

Model-View-Controller

One distinct advantage Swing has over AWT is its employment of the Model-View-Controller (MVC) design pattern (see Figure 3). This design pattern, employed when using object-oriented techniques with user interfaces, was first introduced with Smalltalk. In MVC there are three objects: the model, view and controller. The model has nothing to do with the presentation of data, but with the essential data attributes and their associated values. It notifies views interested in the model when data values change. The presentation or appearance of the user interface is the view object - this

Certicom

www.certicom.com

ensures that the presentation of the model is accurate. As the view object receives notification that a model has changed, it updates the appearance. Any number of views may be associated with a model. For example, one model may be displayed with views as a pie chart, graph or spreadsheet. The controller object manages how the user interface responds to user interactions, such as a button press or movement of a mouse.

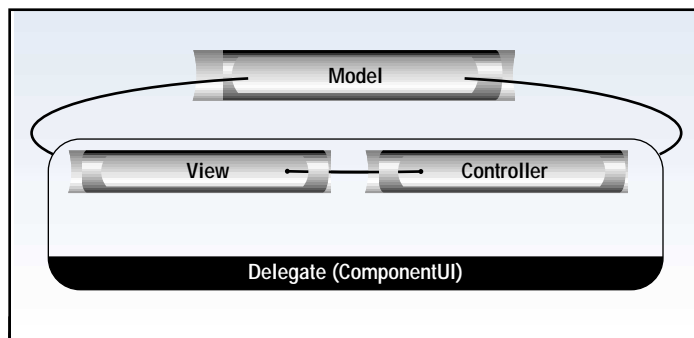


FIGURE 3 The Model-View-Controller design pattern as used in Swing

The view and controller objects play less of a role in Web-based applications than in typical desktop user interfaces. The view object is less significant because most Web-based clients must poll for the current state of the model. Except rarely, the model object doesn't notify the view object when a value has changed. The controller object is less significant because Web-based applications have less user interaction than their desktop counterparts. Accordingly, Swing fuses the view and controller objects into a delegate object, a design called a *separable model architecture*. The delegate object both reacts to user interaction and presents the model. Each component's delegate is derived from the ComponentUI class. For example, the JButton component uses a ButtonUI class and the JLabel component uses a LabelUI class, both of which are derived from the ComponentUI class. These are known as the *pluggable* look and feel interfaces (see Figure 4). Methods in this class – e.g., paint, updateUI – deal with the appearance of a component.

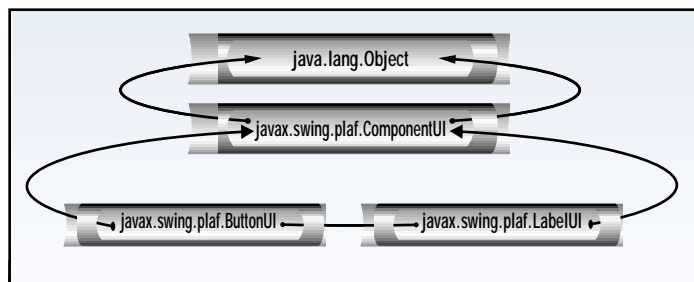


FIGURE 4 Inheritance diagram for Swing pluggable look and feel interfaces

Description of a Self-Contained Client Applet

To illustrate the concept of a self-contained client applet we're going to use the example of a building-monitoring system. This is a system for monitoring the security, temperature and power within a building. The server receives events from telemetry equipment, records them, correlates them when possible and reports them to the building-monitoring clients registered with the server.

When the monitoring client applet is downloaded and started, the main monitor menu is displayed in the browser. The menu contains items such as file, edit, view and help. A "peel-off" menu created from the main menu causes the applet to peel itself from the browser and act as a stand-alone window. Users may perform various actions that create windows within the application – e.g., they can customize the monitoring client to view particular areas of a building or the entire building. A panel containing an event list displays each room or section of the building.

The room name and time of the latest event is displayed at the top of each panel. Events are color-coded based on their importance. If, for

example, the room temperature were configured at 70 degrees and the temperature rose to 75 degrees, a minor event would be generated and it would appear as a yellow button within the list. But if the temperature were to rise above 80 degrees it would become a major event and would appear as a red button. To help them understand the events, users can click on any one of these buttons and a separate detail window will pop up.

Users can acknowledge the event to indicate they're currently investigating it and can enter notes about events at any time. Once an event has been resolved, the user can clear it. A user can have several windows open simultaneously and navigate between them, or minimize any one of them. The monitoring windows will be updated in real time as events are received from the server, even if the monitoring window isn't the current active window. Even though these actions are performed by an applet running in the confinement of the browser's environment, the behavior appears as though it's a typical multiwindowing client/server application executing outside the browser. All GUI and communication components required by the application are encapsulated within the downloaded self-contained client applet.

Comparing Applet and JApplet Classes

In lieu of the java.awt.Applet class, the com.sun.java.swing.JApplet class is used for support of Swing components. Extended from java.awt.Applet, the com.sun.java.swing.JApplet class is a heavyweight component since it isn't extended from the JComponent class. One of the differences between the com.sun.java.swing.JApplet class and the java.awt.Applet class is that it supports the JMenuBar Swing component and Swing lightweight components.

Components aren't added directly to a JApplet, as they are to an Applet. Swing heavyweight components use the com.sun.java.swing.JRootPane class to manage their operations. The JRootPane class, the only child of heavyweight components, is obtained by using the heavyweight component's getContentPane method. The JRootPane returned by the getContentPane method is used to add components to the heavyweight container. Components added to lightweight containers are added directly to the container using the add method of the container concerned.

For example, you add a JPanel to a JApplet as follows:

```
public class demoClass extends JApplet {
    public void init() {
        JPanel panelToAdd = new JPanel();
        getContentPane().add(panelToAdd);
    }
}
```

The JApplet class makes use of the BorderLayout, rather than FlowLayout, as the default LayoutManager – as does the Applet class. Just as with adding components, the getRootPane method is used to set JApplet's layout manager. For example, to set a JApplet's layout manager to BorderLayout:

```
public class demoClass extends JApplet {
    public void init() {
        getContentPane().setLayout(new BorderLayout());
    }
}
```

Creating Menus with Swing Components

One element of our example that runs inside the browser frame is the main menu. This is an excellent example of creating a GUI using components as building blocks. A menu bar, created by using the JMenuBar class extended from the JComponent class, consists of any number of JMenuItem objects, each one with any number of pull-down menu items that can be selected with either a mouse click or a keyboard shortcut.

The JMenuItem class provides methods for creating menus that contain menu items. The JMenuItem class provides methods for creating menu items contained within a JMenuItem. JMenuItem objects allow users to

YouCentric

www.youcentric.com/nobrainier

request program functionality by being selected. In the example application, the selection of the “view as a separate window” JMenuItem causes a main menu to be peeled off the browser and appear outside the browser’s frame.

JMenuItems may contain a text label or an icon. A number of constructors exist for the JMenuItem class, permitting the specification of a label or icon. The default constructor may be used to create a JMenuItem at one point in the code, then set the label later by using the setText method. The example code will create a fully populated JMenuBar, starting with the creation of JMenuItem items with a label:

```
JMenuItem newMenu = new JMenuItem("New");
JMenuItem openMenu = new JMenuItem("Open");
JMenuItem saveMenu = new JMenuItem("Save");
```

The JMenu class has three constructors. A JMenu may be instantiated (1) with no label, using the JMenu default constructor, (2) with a label using the JMenu constructor with a String or (3) with a label and boolean variable indicating the state of the JMenu. A number of JMenu classes will be instantiated here with a label. These will contain the JMenuItem items created above.

```
JMenu fileMenu = new JMenu("File");
JMenu editMenu = new JMenu("Edit");
JMenu toolsMenu = new JMenu("Tools");
JMenu helpMenu = new JMenu("Help");
```

JMenuItem items are added to a JMenu with the add method:

```
fileMenu.add(newMenu);
fileMenu.add(openMenu);
fileMenu.add(saveMenu);
```

Finally, a JMenuBar will be created to contain the JMenus. The JMenuBar has one constructor, the default constructor:

```
JMenuBar menuBar = new JMenuBar();
```

JMenus are added to the JMenuBar with the add method:

```
menuBar.add(fileMenu);
menuBar.add(editMenu);
menuBar.add(helpMenu);
menuBar.add(toolsMenu);
```

The Observer Design Pattern and Events in Java

JDK 1.1 introduced the delegation-based event-handling mechanism. Also known as the publish-subscribe model, it’s based on the Observer design pattern, used when consistency is to be maintained between objects. Java implemented the Observer design pattern by creating Subject and Observer objects. The Subject maintains a list of Observers that are notified when the Subject’s state changes. This set may contain one or more Observers. Observers provide an interface through which the Subject notifies them of a state change. As the Observers are notified of this change, they can examine the Subject to determine how they should behave. In this scenario there’s very low coupling between the Subject and Observer objects since the Subject is unaware what information the Observer object needs or how it behaves.

In Java, Subjects must create an event extended from the java.util.EventObject. In the example application the Subject is a JMenuItem that creates an ActionEvent derived from EventObject. Observers or listeners must implement the java.util.EventListener interface. In this example the actionPerformed method of the ActionListener interface, extended from the EventListener, is implemented.

The addActionListener method is used to register an Observer or listener with a Subject or JMenuItem. If the listener were contained in a class other than the class containing the JMenuItem, an instance of the

class would be created, then registered with the JMenuItem by using the addActionListener method. In our example the class containing the listener is the same class that contains the JMenuItem:

```
openMenu.addActionListener(this);
```

Defining the actionPerformed method of the ActionListener interface specifies an Observer or listener. This method is executed when the ActionListener receives an action event generated by the Subject. One listener can be associated with any number of JMenuItem items. The ActionListener class inherits the getSource method from the EventObject class. This method returns the object that caused the event. In this case it’s used to determine which JMenuItem caused the action event.

```
public void actionPerformed(ActionEvent e) {
    if(e.getSource() == openMenu) {
        // handle action event for openMenu JMenuItem
    } else if(e.getSource() == peelMenu) {
        // handle action event for peelMenu JMenuItem
    }
}
```

Swing ToolTips

It’s often helpful to provide users with more information about a menu item than the label in that item itself. All Swing lightweight components inherit JComponent’s setToolTip method. This method may be used to specify text that appears when the mouse remains over a Swing component for a given period of time, helping users know what the component is for. In our example we associate a tool tip with a JMenuItem, but a tool tip may be associated with any Swing lightweight component.

```
newMenu.setToolTipText("Used to Create Something New");
```

Creating a Window Outside the Browser

Both JFrame and JApplet are heavyweight containers that can be used in very similar ways. The JFrame container can be used to create a window that’s not visually contained within the browser. As a heavyweight container, the JFrame will appear differently on different platforms. The most noticeable difference between platforms will be the appearance of buttons used to minimize, maximize and close the JFrame.

From the user’s point of view, once the JFrame is created, the applet appears to be running independently of the browser. In reality, it continues to execute from within the browser and is restricted to behaving like any other applet. Any connections the applet has to a server are still present in a JFrame appearing outside the browser’s window. Once a window is created outside the browser’s window, users can visit other Web pages or minimize the browser.

In Listing 1, if the peelMenu JMenuItem is selected, the event handler for the peelMenu will cause a main menu to appear – not within the JApplet, but within a JFrame outside the browser window. The steps to perform this task are identical to those you take to perform it within a JApplet, with the exception of having first to create a JFrame. The java.awt.Window method pack is called to lay out and position all components contained in the Window at their proper size.

Description of Event Panel

A class that implements the ListSelectionListener interface is used to contain each event panel. The valueChanged method in this interface is provided to handle activity when a JList selection value changes. A ListSelectionEvent object is received by the valueChanged method, which can be queried to determine more information about the selection list that generated the event. The Swing JList, a component used to create a multiple-selection list, is used to display events occurring in each building section being monitored. The JList is created, then associated with a renderer with the following calls:

Zucotto

www.zucotto.com

```

class EventPanel extends JPanel implements ListSelectionListener {
public EventPanel() {
    JList list = new JList();
    list.setCellRenderer(new ARenderer());
}
public void valueChanged(ListSelectEvent event) {
    // handle selection list value change
}
}

```

Customizations to the JList include the specification of a horizontal and vertical scroll pane, as needed:

```

JScrollPane scrollPane = new JScrollPane(list,
ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED,
ScrollPaneConstants.HORIZONTAL_SCROLLBAR_AS_NEEDED);

```

After all customizations are made to the JList, it's added to a JPanel that also contains JLabels for the identification of the building section (see Figure 5).



FIGURE 5 Windows created by a self-contained client applet running outside the browser

Cell Rendering

The ListCellRenderer interface is used to specify how a component represents data in a list. The ListCellRenderer has one method, getListCellRendererComponent, that appropriately displays a data value. This method receives the object being represented in the list and returns a component used for rendering the item. This method is invoked each time the cell is required to be repainted, such as when a window is resized. Time-consuming operations, such as retrieving data from a server or expensive GUI operations, shouldn't be placed in the getListCellRendererComponent method. Server-side data may be cached in the client for use by the getListCellRendererComponent method. A constructor for the class implementing the ListCellRenderer interface may be used to perform time-consuming operations or operations that need occur only once. To avoid performance degradation, for example, the constructor of the ARenderer class in Listing 2 creates the borders and fonts when the class is instantiated.

The method should contain only operations required to dynamically change the appearance of the data being represented. For example, the renderer may visually represent an object in different forms based on the events and data it receives from a server. The getListCellRendererComponent method can be used to translate an integer value into text.

Another example: if an attribute of the object exceeds a certain value, the background of the label can be set to a certain color (see Listing 2).

Summary

Java Swing introduces a new GUI architecture supported by a host of GUI components that can be used to create effective GUI tools that look and behave similarly to client/server windowing applications. Long-running applets can use these components to provide users with windows outside the browser's frame, permitting them to visit other Web pages and thus making the use of the long-running applets more productive. 🍌

AUTHOR BIOS

Thomas Czernik is a member of AT&T's technical staff with 18 years of systems development experience. Tom holds an MS in computer science.

Rolf Kamp is a member of AT&T's technical staff where he develops network operation software. An adjunct faculty member at Brookdale Community College, Rolf holds an MS in computer science and an MBA.

czernik@att.com

rfk@att.com

Listing 1

```

JFrame menuFrame = new JFrame("MONITOR MENU");
NotifyMenu mainMenu = new NotifyMenu();
JPanel pPanel = new JPanel();
JLabel lLabel = new JLabel("MONITOR MENU");
Font lFont = new Font("Serif",Font.BOLD|Font.ITALIC,24);
lLabel.setFont(lFont);
lLabel.setForeground(Color.white);
pPanel.setBackground(Color.blue);
pPanel.add(lLabel);
menuFrame.setJMenuBar(mainMenu.getMenuBar());
menuFrame.getContentPane().add(pp);
menuFrame.pack();
menuFrame.show();

```

Listing 2

```

class ARenderer extends JLabel implements ListCellRenderer {
    static Font f = new Font("Helvetica",Font.BOLD,10);
    static SoftBevelBorder sbb = new SoftBevelBorder(SoftBevelBorder.RAISED);
    static CompoundBorder cb = new CompoundBorder(new MatteBorder(2,2,2,2,Color.green),new SoftBevelBorder(SoftBevelBorder.RAISED));

    public ARenderer() {
        setOpaque(true);
        setFont(f);
        setForeground(Color.black);
    }

    public Component getListCellRendererComponent(JList l,
        Object v, int i, boolean isSelected, boolean cellHasFocus) {
        LocalEvent a = (LocalEvent)v;
        setToolTipText(a.description);
        setText(a.symbol);
        if (a.severity == 0) {
            setBackground(Color.gray);
        } else if (a.severity == 1) {
            setBackground(Color.cyan);
        } else {
            setBackground(Color.white);
        }
    }

    // additional customization code...
    return this;
}

```



Codemarket

www.codemarket.net/specialoffer

PowerTier 6's PowerPage

Use this latest version to instantly Web-enable your EJBs!

WRITTEN BY
JASON WESTRA



Persistence Software, Inc., recently released the latest version of its EJB server, PowerTier 6. It's a little different from your run-of-the-mill EJB servers, though. This JavaOne 2000 special-edition issue of EJB Home will enlighten you about PowerPage, a hot feature that will put PowerTier 6 on every EJB evaluator's radar!

Let's first understand why PowerTier 6 and PowerPage are such key ingredients in any large-scale, e-commerce solution today.

The E-Commerce Dilemma

My company has worked with numerous dot-coms this year, and most have puzzled over the same dilemma: How do we deliver a scalable, Web-enabled application, using standards-based technologies, in "Internet time"?

Four key items need to be addressed in what I call the *e-commerce dilemma*:

1. Scalability
2. Web enabling
3. Standards-based technologies
4. Internet time (time-to-market)

Tackling an e-commerce application generally involves give or take among these four items. For instance, maybe you decide to build a scalable, standards-based application, but it'll take an extra two months to get it fully functional and performance tested. Or maybe you opt for scalability and time-to-market, and choose a proprietary solution to reach your product deadline successfully. How do you address all four concerns and still make an e-commerce application in record time? There's an answer to this dilemma, and it's called PowerTier 6.

PowerTier 6 to the Rescue

Two critical elements are key to solving the e-commerce dilemma:

- *An EJB server for deployment of e-commerce components*
- *Model-driven development and code generation*

PowerTier 6 is a feature-packed application server that delivers both elements to solve the four key items of this dilemma.

MEETING THE SCALABILITY CRITERION

Building on top of its original patented caching technology for scalability, the PowerTier 6 EJB server meets the scalability demands that e-commerce application and high-traffic sites require. Around 90% of the traffic on the Web is browsing (i.e., read only). PowerTier's shared cache provides a means to read EJB data at in-memory speed rather than disk speed. PowerTier 6 caches often-browsed data, improving the speed at which useful information is returned to the consumer. Timeliness enhances users' experience and prevents them from surfing to another site while waiting for responses from your Web application.

PowerTier's EJB server not only takes advantage of its patented caching, it also supports load balancing and fault tolerance of servers and EJB components.

MEETING THE WEB-ENABLED CRITERION

Persistence Software recently acquired 10BaseJ and incorporated its ServletMill product into the PowerTier 6 application server. ServletMill is written entirely in Java and includes support for Java Servlet API 2.1 and a JavaServer Pages (JSP) engine that is compliant with Sun's JSP specification 1.0.

The PowerTier servlet engine, like its EJB server, is highly scalable and fault tolerant. It can run a servlet in several "zones," which are essentially clusters of servlets. Zoning provides scalability by routing user activity across zones, ensuring that no single servlet engine is overloaded with requests. Also, ServletMill can fail over the state of a user's HttpSession. Fault tolerance in the PowerTier servlet engine is implemented by saving additions or updates to sessions on a shared file system. When a zoned servlet fails, another servlet retrieves

the session state and continues processing the request.

The PowerTier servlet engine enables the dynamic content from JSPs and servlets to be displayed in HTML form for your Internet application needs. For static content PowerTier 6 bundles Apache Web server and includes a plug-in for Microsoft IIS.

PowerPage, PowerTier's innovative JSP generation feature, creates JSPs that run in the PowerTier servlet engine.

MEETING THE STANDARDS-BASED TECHNOLOGIES CRITERION

PowerTier 6 is an application server built on standard technologies such as EJBs, JSPs and RMI-IIOP. Persistence is also a J2EE (Java 2 Enterprise Edition) licensee, further guaranteeing its loyalty to standards. By building your e-commerce application with PowerTier 6, you'll be safely betting your development and deployment environment on industry-accepted standards, which will reduce the cost and complexity of maintaining your e-commerce solutions over time.

MEETING THE "INTERNET TIME-TO-MARKET" CRITERION: INTRODUCING POWERPAGE

PowerTier 6 is one of the few application servers on the market to offer built-in support and third-party add-ons to ease your fledgling team into *n*-tiered EJB development. PowerTier's Object Builder offers model-driven code generation for EJBs; with release 6 it also incorporates a generation capability from the model to JSPs called PowerPage.

PowerPage automatically makes your PowerTier entity beans accessible from JSPs by generating all code necessary for a browser-based application with the click of a button. It allows instant Web-access to a scalable, standards-based back end, allowing your

Verge Technologies Group Inc

www.ejip.com

team to finish your product in Internet time without worries of missed deadlines, supporting growth or losing time-to-market.

I've briefly described how PowerTier 6 provides the critical components for solving the e-commerce dilemma. For the remainder of this article I'll focus on PowerPage and its feature set to help you understand the role it can play in your development efforts.

PowerPage — RAD for JSPs

The PowerPage rapid application development (RAD) process follows the same philosophy as PowerTier's EJB development process: a single object model should drive development. Model-driven development prevents you from having to make multiple code modifications to EJBs and JSPs in support of a single business domain change.

The PowerTier EJB and PowerPage development process starts when you design an object model with PowerTier's built-in Object Builder or with industry-recognized tools like Rational Rose and TogetherJ. Afterwards, you generate your EJBs representing business entities in the model and JSPs that represent the front end of your Web application. Figure 1 details how the design and development process with PowerTier 6 is tightly coupled into a single, easy-to-use toolset.

TASK	TRADITIONAL EJB SERVER	POWERTIER 6 WITH POWERPAGE
Create entity beans	Hand-code	Automatically generated
Create database schema	Hand-code	Automatically generated DDL
Create JSPs or servlets	Hand-code	Automatically generated
Create JSP's nonvisual JavaBeans	Hand-code	Automatically generated
• String to data type conversion		
• Data type validation		
• EJB Home lookups		
• EJB Remote Method Invocations		
• Etc.		

TABLE 1 Steps to Web-enable your EJBs

PowerTier's patented generation technology, PowerTier Object Builder for PowerPage, can take multiple inputs to drive the development of your EJBs and JSPs. You can use a third-party modeling tool or a database schema from top industry RDBMS vendors for input, or you can enter your model manually into Object Builder. Output from Object Builder for PowerPage is a Web-enabled JSP application backed by PowerTier's renowned, scalable, EJB architecture.

Support for Iterative Development and Customization

Iterative development is commonplace in today's software development,

and Object Builder provides iterative support through code insertion points that represent areas within generated code into which you can safely enter custom logic. When you make a change to your model and regenerate your EJBs and JSPs, you won't lose your custom logic. Numerous RAD tools on the market today don't offer this option. They're one-stop shops that simply provide a jump start through code generation but no iterative capabilities throughout the life cycle of development.

You can choose to regenerate only JSPs or only EJBs, a nice option when you already have an Object Builder EJB model and want to quickly generate a JSP front end to show a functional prototype to management.

In addition, you can customize the look-and-feel of your generated JSPs. PowerPage generates what it refers to as a Project.css file, which is a cascading stylesheet for your HTML presentation. This file can be modified to customize the appearance of your application. You may change presentation characteristics like background color and font for your project to give it a unique look-and-feel.

Steps to Web-Enabling EJBs

Those of you who are avid readers of **EJB Home** have heard this message from me before: "EJB development is not easy!" This holds true in the context of a traditional EJB server; however, PowerTier 6 with PowerPage has made me eat my words to some extent. The methodology and technology that PowerTier uses to build a functional, Web-enabled EJB application is dramatically easier than any I have worked with since EJB servers were first available in 1998.

Table 1 lists the tasks involved in creating an EJB application and Web-enabling it with Java servlets or JSPs. As

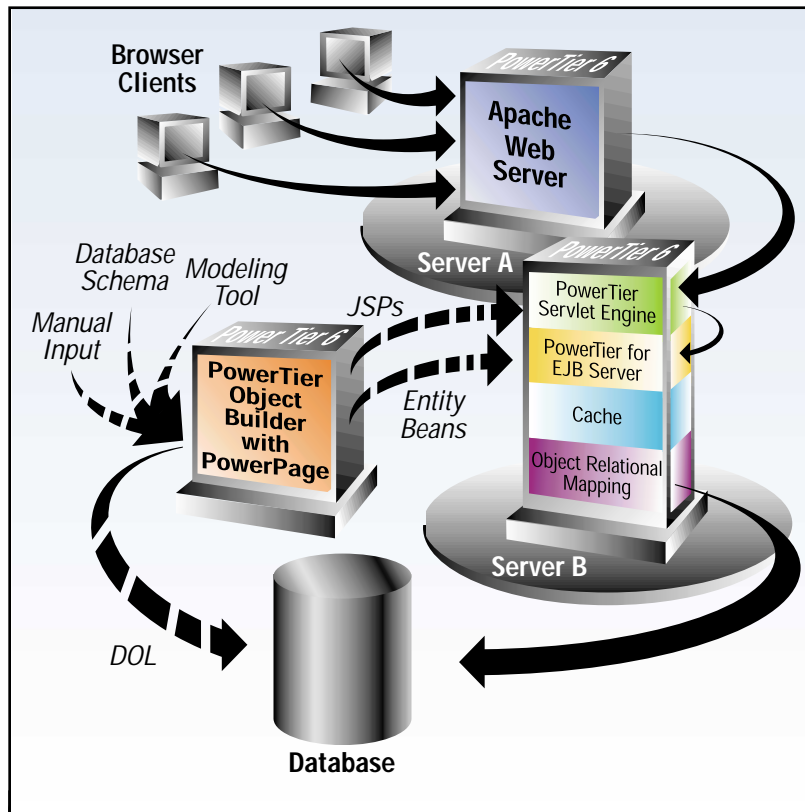


FIGURE 1 Creating JSPs and entity beans with PowerPage

Amazon.com

www.amazon.com

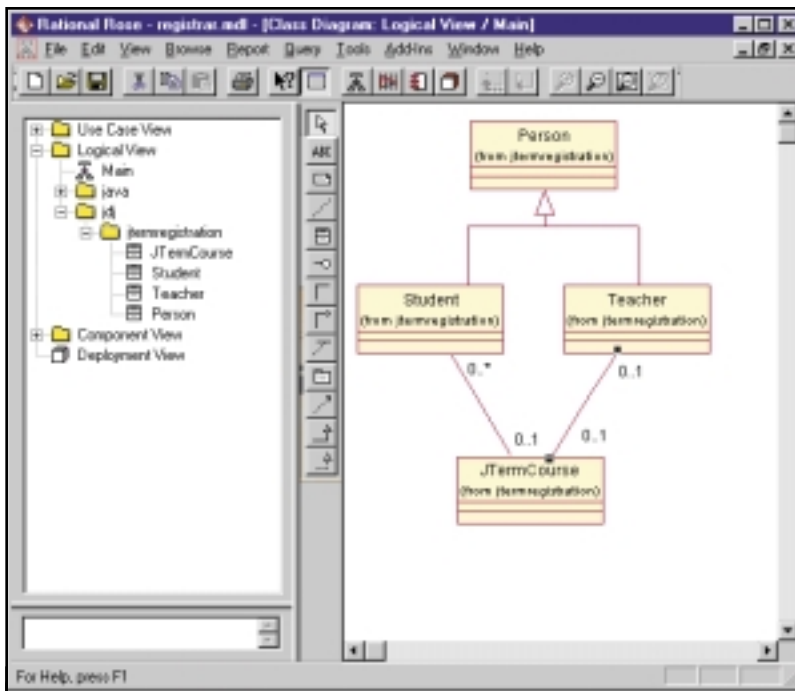


FIGURE 2 UML model of J-Term registration application

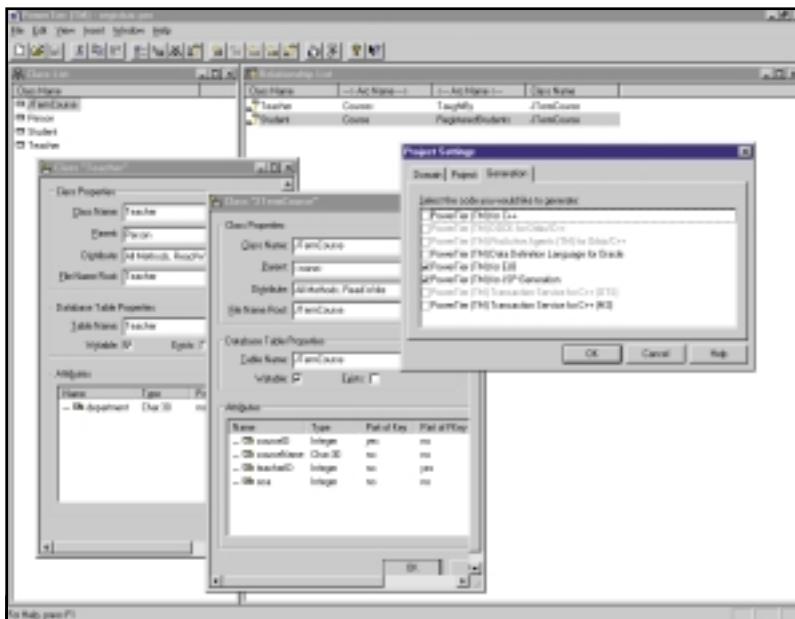


FIGURE 3 PowerTier Object Builder

you can see, PowerTier 6 does the main tasks for you through seamless generation; with traditional EJB servers you have to perform each task manually.

Example Use of PowerPage

When I first heard about PowerPage, I was eager to dig into the technology myself. I don't normally evaluate something with the vendor-provided examples. Instead, I create something from scratch to test the process on my own. I'll walk you through the process I followed and provide you with insights into the PowerPage product along the way.

J-Term Registration

J-Term stands for *January Term*, which is a monthlong term in some colleges during which students take only one course and professors teach only one course. To investigate the PowerPage product, I built a simple J-Term Registration application that allows the entry of J-Term courses, teachers and students. The UML model for the application is as shown in Figure 2. I used inheritance and associations to show PowerTier's ability to generate different relationships between EJBs and map them to database tables and JSP pages.

First I manually entered the classes and relationships into the PowerTier Object Builder (I could have used a third-party tool for this step). After Object Builder contained my model, I marked two checkboxes to indicate I wanted to generate both PowerTier EJB and JSP files (see Figure 3). Then I selected the File -> Generate Code menu item and Object Builder's generator created both my EJBs and JSPs for the J-Term Registration example.

With my J-Term EJBs, JSP files and HTML files generated (see Figure 4), I could have modified the stylesheet to my liking - added custom code inside code insertion points, or more JSP scripting - but chose not to for this example. Instead, I deployed the J-Term Registration example as is into the PowerTier 6 server and its bundled Apache Web server, then started the PowerTier server to test my auto-generated application. My application functioned with simple CRUD (Create, Read, Update, Delete) functionality!

I didn't get a chance to test round-trip engineering with PowerPage generation. I'm not sure how easy it is to add custom code to a JSP or generated nonvisual JavaBean, but my guess is that it's just as easy as adding code to generated EJBs.

PowerPage Limitations

PowerPage accesses entity beans directly. This design conflicts with the generally accepted entity bean wrapper pattern. In this design pattern entity beans aren't accessed directly from clients. Instead, session beans are used to manage transaction semantics and manipulate multiple entities in a single business function.

I see PowerPage as a powerful feature for developing Web-enabled e-commerce applications and functional prototypes. However, I don't believe it will add as much value to large-scale enterprise efforts with disparate clients. These applications require business processes to be modeled as session beans in order to encapsulate business logic and transaction boundaries for multiple client platforms, not just browser clients. I look forward to future product releases of PowerTier to see if Persistence tackles this market with a product as well engineered as PowerPage.

I discovered some bugs in the initial release as I was working with it, but was not able to track the cause to PowerPage or the newly integrated servlet engine before the article deadline. Integrating a servlet engine and creating a new JSP feature in one release is a daunting task to accomplish for any engineering

Elixir Technology

www.elixirtech.com/download

group. Nevertheless, PowerTier 6 is an application server with a rich feature set.

Conclusion

PowerPage provides a bridge between application server components and Web

pages – automating EJB access from Web pages and eliminating tedious manual coding. Together, PowerPage and PowerTier Object Builder ensure faster deployments by reducing development time and increasing reliability and efficiency of code.

The ability to preview application functionality at any point in the development process means better user interfaces, more reliable products and shorter project cycles. The underlying PowerTier application server guarantees the rapid response that will keep Internet customers coming.

While in Silicon Valley last March, I had a chance to meet with Chris Keene, CEO of Persistence, as well as their VP of engineering and the senior engineer who developed PowerPage. My impression of Persistence as a company is that it is dedicated to providing end-to-end solutions that eliminate the learning curve for *n*-tiered, enterprise Java development. This commitment is evident in its acquisition of 10BaseJ to bundle a scalable servlet/JSP engine in PowerTier 6 as well as the engineering team's innovative approach toward Web-enabling EJBs with PowerPage.

Reference

“PowerTier PowerPage: Instant Web Access to Enterprise JavaBeans” – Contact Persistence Software, Inc., at www.persistence.com, to have a copy of the technical white paper sent to you. ☛

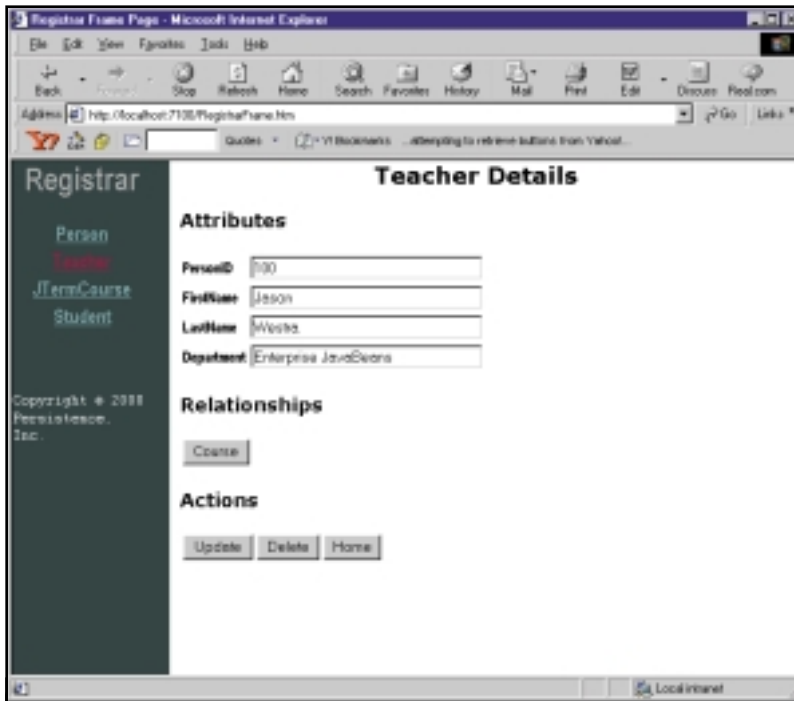


FIGURE 3 HTML from the PowerPage-generated JSP

AUTHOR BIO

Jason Westra is the CTO of Verge Technologies Group Inc., a Java consulting firm specializing in e-business solutions with Enterprise JavaBeans.

westra@sys-con.com

SYS-CON Radio

www.sys-con.com

?

Your Enterprise JavaBeans will probably manage dependent objects. You need to save these objects – cleanly and efficiently – in a relational database.

This article tells you how.

WRITTEN BY DANIEL O'CONNOR

Say you're writing an Enterprise JavaBean that represents a persistent object, such as a customer or a product. You have two choices for getting data (such as customer name and product number) from the bean to the database and back:

- You can let the bean's runtime environment – its container, in EJB speak – do the heavy lifting for you...
- ...or you can provide the logic yourself along with your bean.

It seems like an easy choice. Why write code when you don't need to? Frequently, in fact, container-managed persistence will be a good match for a project. However, if you want your bean to be portable across multiple EJB servers, or if you find that the container-managed persistence provided by your chosen EJB server is inadequate, you'll need to turn to bean-managed persistence.

According to the current version of the Enterprise JavaBeans Specification (v1.1), a compliant EJB container isn't required to provide any support for mapping the container-managed fields to a database schema. For instance, it could use Java serialization to save the beans to a file. Most if not all commercial and open-source containers will map fields to table columns in a database. But some will do it better than others. You'll have trouble with some products, for example, if you want to map your bean to multiple rows in multiple tables.

Bean-Managed Persistence Using a Proxy List

The Middleware Company

www.middleware-company.com/info

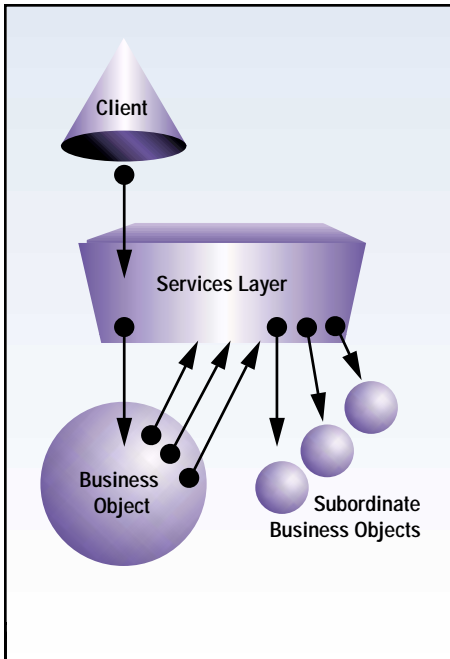


FIGURE 1 Bean-to-bean method calls go through a services layer.

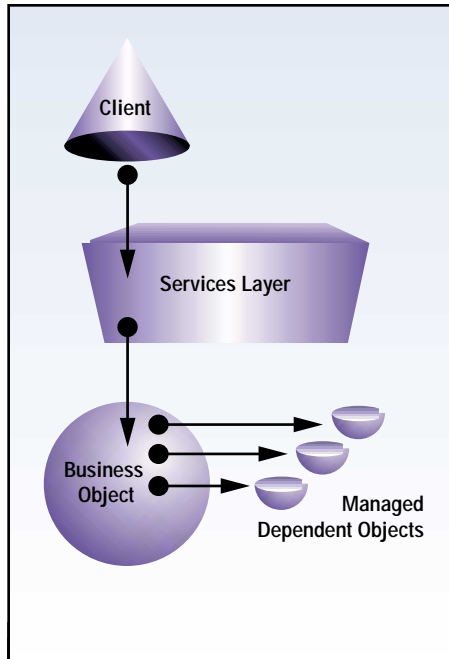


FIGURE 2 Bean-to-dependent object calls are lighter weight.

DATA	SQL SYNTAX
New	Insert
Changed	Update
Discarded	Delete
Unmodified	

FIGURE 3 Beans should perform updates the way a relational database expects.

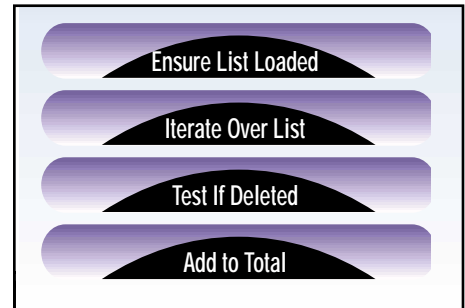


FIGURE 4 A naive implementation mixes business logic with persistence logic.

In fact, you probably do want to map your bean to multiple rows in multiple tables. Your beans should be coarse-grained, managing their own dependent objects. Your order bean should have line items implemented as a helper class, rather than as references to line-item EJBs. Your customer bean should have addresses rather than references to address beans. The overhead of bean-to-bean calls between an order and its line items, or a customer and its addresses, would be prohibitive. Your EJB server can provide you with many services, such as declarative transactions, security, and even load balancing and failover. But there's no free lunch, and the price you pay is indirection. Every call you make goes through a layer whose purpose is to provide these services (see Figure 1). Managing dependent objects reduces the frequency of trips through this indirection layer (see Figure 2). And a bean with multiple dependent objects needs to be stored in multiple tables, in multiple rows, if you want to maintain a normalized schema.

Technically, bean-managed persistence doesn't mean you have to write your own database access code. It just means the bean, rather than the container, provides the persistence logic. Several good object-relational mapping tools are on the market, and they can be portable from server to server along with your bean. But you may find it impossible to use these tools because of cost (some have runtime fees and/or a hefty per-developer price tag), distribution practicalities or reasons of your own. This article will tell you what you need to do to write your own persistence for a coarse-grained entity bean.

Requirements for Dependent Objects

An efficient implementation of bean-managed persistence for dependent objects will have two features:

- Load-on-demand
- Partitioned storage logic

Load-on-demand means that the dependent objects aren't loaded until they're actually needed. The EJB framework will call a function in your entity bean to indicate that persistent data should be made available to the bean's business logic. The bean could load the dependent data at this point. But if the business logic doesn't make use of certain dependent data during the current transaction, that database access was wasted. For instance, changing a customer's credit rating may not require access to any address, so the addresses shouldn't be loaded. If the

dependent data is accessed, it can be loaded at that time. (This is also known as *lazy loading*.) Partitioned storage logic is necessary so that the bean updates the database the way a relational database expects: new data is inserted, changed data is updated, discarded data is deleted and unchanged data is left alone (see Figure 3). The alternative – wiping out the records and reinserting them – is too horrible even to contemplate.

A Good Idiom

To implement load-on-demand, you could scatter calls throughout your business logic to functions with names like “ensureAddressListLoaded” and “ensureLineItemListLoaded” – that is, if you want to be the poster child for ugly code. And to store your dependent objects to the database, you could have each object keep track of its status: NEW, UPDATED, DELETED or UNMODIFIED. As you totaled the line items in a purchase order, you'd need to check each object to see if it had been deleted (see Figure 4). Don't forget, or you're going to have some unhappy customers.

A better idiom is to group the logic related to persistence with a collection class. Your business logic for an order probably works with a list of line items. To delete an object, the most natural thing to do is probably to remove it from the list. To add an object, the most natural thing is to append it to the list. And simply calling a method on the list should be the signal to your persistence logic that the items in the list need to be loaded from the database. If you use a smart list that knows how to do these things, nothing else needs to be done from the perspective of the business logic programmer. In the example of totaling line items in a purchase order, you'd simply iterate through the objects in the list (see Figure 5).



FIGURE 5 With the persistence logic in the collection class, the business logic is simpler to code and maintain.

Elixir Technology

www.elixirtech.com/download

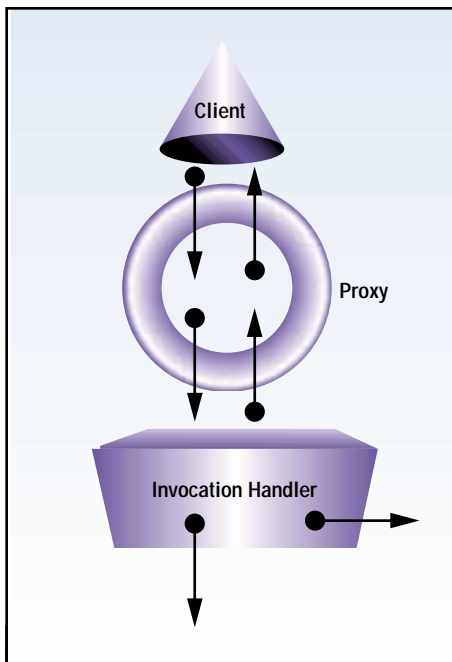


FIGURE 6 The Proxy/InvocationHandler combination is a flexible tool.

“If you need to roll your own persistence for EJBs, there’s no better place for the persistence logic than a collection class”

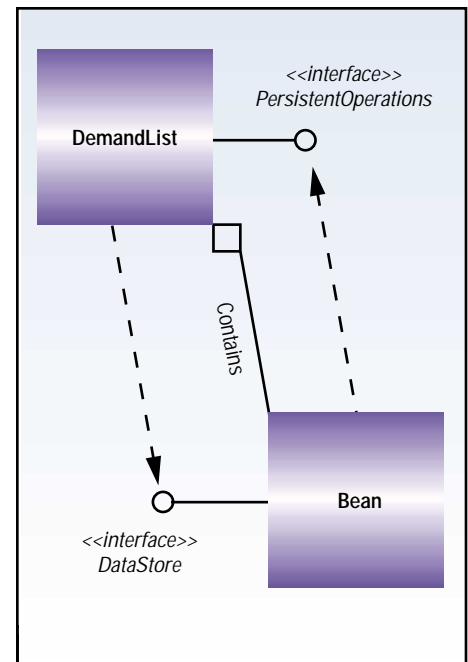


FIGURE 7 The smart list uses two interfaces to manage its persistence.

Behind the scenes, a smart list implementation is keeping track of an “isLoaded” variable. When the list is accessed, it checks this variable first to see if the data needs to be loaded from the database. If so, it loads it. It keeps a set of references to all the objects it loads to distinguish them from new objects added to the list. If an object is removed from the list, it’s added to an internal list of deleted objects. This deleted objects list is used by the persistence logic, but not, typically, by the business logic programmer.

The Proxy List

The smart list needs a layer of indirection between the list interface and the actual list storage. At this layer a method call on the list to remove objects will first add those objects to the internal deleted objects list. Also, a call to any function will check to see if the list data has been loaded. One possible way to gain this indirection is simply to implement the `java.util.List` interface and delegate the calls to a private internal “backing” list. A more elegant way is to use the `java.lang.reflect.Proxy` class newly available in JDK 1.3. (Note: Using the Proxy class limits your bean’s portability to servers that support JDK 1.3. All the techniques and code discussed in this article are easily adapted to the “delegation list” compatible with earlier JDKs.)

The Proxy class dynamically creates an implementation of an interface that will automatically forward all its calls to a middle layer called `InvocationHandler` (also in package `java.lang.reflect`). In a subclass of `InvocationHandler` you can forward the method call (or not), take action before or after forwarding it, alter its parameters and change the returned object (see Figure 6). As you can see, this is more than enough functionality to implement our smart list. The uses for this Proxy are many: it can be used to handle user interface events and to provide a “poor man’s multiple inheritance,” and has even been used to implement an open-source EJB server, EJBoss. (For more information on the Proxy and `InvocationHandler` classes, see the article at <http://java.sun.com/products/jfc/tsc/articles/generic-listener2/index.html> on Sun’s Web site.)

Take a look at `ListInvocationHandler` (see Listing 1). It’s the smart list implementation that keeps track of deleted objects, the set of original objects and whether the data has been loaded from the database. It also takes as a constructor parameter its “backing” list so that any class implementing the List interface (`LinkedList` or `ArrayList`) can be used, depending on how the data is typically accessed. The main functionality

is in the `invoke` method. Here I check to make sure the data has been loaded from the database. I also check for any List function that removes an object so I can make a copy to use for calling “delete” later on the database. An important point: several List functions will return a reference to the backing list unless these too are interposed on. Any method that returns an `Iterator` (which points to the backing list) must instead be made to return an `Iterator` pointing to the interposed list. I did this using – you guessed it – another Proxy (see Listing 2). Any method that returns a `Collection` must either be interposed on or made unmodifiable.

Persistence Details

My smart list implementation uses two persistence-specific interfaces (see Figure 7). The first, `PersistentOperations` (see Listing 3), is implemented by the smart list itself (in `ListInvocationHandler`). It’s for list operations needed by the bean’s persistence plumbing, rather than the business logic. You can get a list of deleted objects to actually delete them. You can get the set of original objects to decide between insert and update operations. You can add an object to the list so that it’s tagged as an original object rather than a new one. You can tell the list that the bean has been asked to save itself to persistent storage, and to take whatever actions are necessary. For symmetry more than need, you can also ask the list to load itself from persistent storage. Typically, you’ll let the list decide this for itself.

The second persistence-specific interface used by the list is `DataStore` (see Listing 4). Although your business logic can treat the smart list as a regular list, you need to put the SQL somewhere, and this interface is the gateway to that “somewhere.” Your bean will pass an implementation of this interface to the smart list factory (`DemandListFactory`; see Listing 5). When the list needs to save or restore its data, it will call methods in this interface, passing a reference to its `PersistentOperations` interface.

Implementation Example

A simplified example of using a proxy list with an EJB with bean-managed persistence is available on the *Java Developer’s Journal* Web site. The bean implements a customer-has-line-items model. A customer has an ID and a name, and an unlimited number of line items of products he or she has ordered. In order to give the smart list a workout, I’ve written functions so the business logic programmer can add, delete or change a

JavaOne Conference

<http://java.sun.com/javaone>

line item, and use the entire set of line items at once. The list is initialized when the bean is created (a new customer record is inserted) or loaded (an existing customer record is read from the database). An anonymous DataStore implementation is passed to the DemandListFactory, which will call bean methods whenever DataStore methods are called by the list.

One bean method will do a simple select on the database table where line items are stored. As it iterates through the result set, it calls the PersistentOperations's addFromStore method, which will indicate to the list that the object already exists in the database and needs to be updated, not inserted, when the list is stored. Another bean method that stores the list is only slightly more complicated. It must use the information available from the PersistentOperations interface to partition the objects into three sets: insertions, updates and deletes. You'll notice that I'm using an isModified function to further partition updates from unmodified instances. It's possible to do this in the smart list as well by keeping a copy of each original object and then comparing it to the object's state just before the database update. There are disadvantages to this technique, however, depending on the memory required to keep copies of those objects and the processing time required to compare object states. In any case, implementing this is beyond the scope of this article.

To test my Customer implementation, I've included a stateless session EJB that will give it a good workout. Fronting an entity EJB with a "business process" session, EJB is a common design pattern. Here it allows us to include multiple adds, deletes, updates and totals within

one transaction. Since the typical EJB server will load and store persistent data on transaction boundaries (load when the transaction begins and store when it ends), this is important for our testing. Obviously, the stateless session EJB isn't a good example of an actual business process.

Conclusion

It isn't difficult to write an efficient, easy-to-use Enterprise JavaBean using bean-managed persistence. Well, what I should really say is that it's not much harder than writing the SQL code that your bean will use. If you need to roll your own persistence for EJBs, there's no better place for the persistence logic than a collection class. In this case we used a list, but the same technique would work for a Set or a Map, or even a tree document, depending on your needs. And if your target market allows you to write to the JDK 1.3, the new Proxy class can increase the expressiveness and effectiveness of your code. ☺

AUTHOR BIO

Daniel O'Connor is an independent software developer writing enterprise management software for the not-for-profit field. He has a BA from Williams College and an MA from SUNY University at Albany.

docodan@nycap.rr.com

Listing 1: ListInvocationHandler.java

```
package orders.demandlist;

import java.lang.reflect.InvocationHandler;
import java.lang.reflect.Method;
import java.util.*;

public class ListInvocationHandler implements
    InvocationHandler, PersistentOperations
{
    private List backingList;
    private DataStore dataStore;
    private boolean dataLoaded = false;
    private List deletedItems = new LinkedList();
    private Set originalSet = new HashSet();

    public ListInvocationHandler( List list,
        DataStore dataStore )
    {
        this.backingList = list;
        this.dataStore = dataStore;
    }

    public Object invoke(Object proxy,
        Method method, Object[] args) throws Throwable
    {
        // persistent operations interface
        if (method.getDeclaringClass().equals(
            orders.demandlist.PersistentOperations.class
        ))
            return method.invoke( this, args );
        else
        {
            // list interface
            if (!dataLoaded)
            {
                dataStore.load( this );
                dataLoaded = true;
            }
            processDeletedItems( method, args );
            Object obj = method.invoke( backingList,
                args );
            if (obj instanceof Iterator)
            {
                return DemandListIteratorFactory.
                    getDemandListIterator( (Iterator)obj,
                        this );
            }
            else if (obj instanceof List)
            {
                return Collections.unmodifiableList(
                    (List)obj );
            }
            else
                return obj;
        }
    }

    // PersistentOperations implementation
    public void loadFromStore()
    {
        dataStore.load( this );
    }

    public void saveToStore()
    {
        if (dataLoaded)
        {
            dataStore.persist( this );
            originalSet.addAll( backingList );
        }
    }

    public List getDeletedObjects()
    {
        return Collections.unmodifiableList(
            deletedItems );
    }

    public Set getOriginalSet()
    {
        return Collections.unmodifiableSet(
            originalSet );
    }

    public List getCurrentList()
    {
        return Collections.unmodifiableList(
            backingList );
    }

    public void addFromStore( Object obj )
    {
        backingList.add( obj );
        originalSet.add( obj );
    }
}
```

Simplex Knowledge Company

www.skc.com

Ajile Systems

www.agile.com

Computer- Work.com

www.computerwork.com

```
// package protected iterator support
void notifyObjectRemoved( Object obj )
{
    deletedItems.add( obj );
}

// implementation
private void processDeletedItems(Method method,
    Object[] args)
{
    String methodName = method.getName();
    if (methodName.equals("clear"))
    {
        deletedItems.addAll( backingList );
    }
    else if (methodName.equals("removeAll"))
    {
        deletedItems.addAll( (Collection)args[0] );
    }
    else if (methodName.equals("retainAll"))
    {
        List tempList = new LinkedList();
        tempList.addAll( backingList );
        tempList.removeAll( (Collection)args[0] );
        deletedItems.addAll( tempList );
    }
    else if (methodName.equals("remove"))
    {
        Class[] paramTypes =
            method.getParameterTypes();

        if (paramTypes[0].equals( Integer.TYPE ))
        {
            Object obj = backingList.get(
                ((Integer)args[0]).intValue() );
            deletedItems.add( obj );
        }
        else
        {
            deletedItems.add( args[0] );
        }
    }
}
}
```

Listing 2: DemandListIteratorFactory.java

```
package orders.demandlist;

import java.util.*;
import java.lang.reflect.InvocationHandler;
import java.lang.reflect.Proxy;
import java.lang.reflect.Method;

public class DemandListIteratorFactory
{
    public static Iterator getDemandListIterator(
        final Iterator iterator,
        final ListInvocationHandler invocationHandler)
    {
        InvocationHandler handler =
            new InvocationHandler()
        {
            private Object cacheLastRetrieval = null;

            public Object invoke(Object proxy,
                Method method, Object[] args)
                throws Throwable
            {
                String methodName = method.getName();
                if (methodName.equals("next") ||
                    methodName.equals("previous") )
                {
                    Object obj = method.invoke(
                        iterator, args );
                    cacheLastRetrieval = obj;
                    return obj;
                }
                else if (methodName.equals("remove"))
            }
        }
    }
}
```

```

    {
        Object obj = method.invoke(
            iterator, args );
        invocationHandler.notifyObjectRemoved(
            cacheLastRetrieval );
        return obj;
    }
    return method.invoke( iterator, args );
}
};

Class clazz = null;
if (iterator instanceof ListIterator)
    clazz = java.util.ListIterator.class;
else
    clazz = java.util.Iterator.class;
return (Iterator) Proxy.newProxyInstance(
    java.util.List.class.getClassLoader(),
    new Class[] { clazz }, handler);
}
}

```

Listing 3: PersistentOperations.java

```

package orders.demandlist;

import java.util.*;

public interface PersistentOperations
{
    public void loadFromStore();
    public void saveToStore();

    public void addFromStore( Object obj );
    public List getDeletedObjects();
    public Set getOriginalSet();
    public List getCurrentList();
}

```

Listing 4: DataStore.java

```

package orders.demandlist;

import java.util.*;

public interface DataStore
{
    public void load(
        PersistentOperations persistOp );
    public void persist(
        PersistentOperations persistOp );
}

```

Listing 5: DemandList Factory.java

```

package orders.demandlist;

import java.util.*;
import java.lang.reflect.InvocationHandler;
import java.lang.reflect.Proxy;

public class DemandListFactory
{
    public static List getDemandList( final List
        list, final DataStore dataStore )
    {
        InvocationHandler handler =
            new ListInvocationHandler(list, dataStore);

        return (List) Proxy.newProxyInstance(
            PersistentOperations.class.getClassLoader(),
            new Class[] { List.class,
                PersistentOperations.class }, handler);
    }
}

```



N-ary

www.n-ary.net

SlangSoft

www.slangsoft.com

SlangSoft

www.slangsoft.com

Monitoring and Diagnostics of CORBA Systems

Demystifying the CORBA communication bus to enable 'distributed debugging'



WRITTEN BY
TODD SCALLAN

Developing distributed applications, in contrast to developing traditional single-process applications, requires a completely different level of monitoring and diagnostic support. In this article I'll discuss how to monitor and diagnose distributed applications based on the CORBA standard.

Perils of Distributed Applications

As organizations make key aspects of their businesses ready and enabled for e-business, multitiered distributed applications are becoming increasingly ubiquitous. Diverse by nature, e-business systems require middleware to integrate middle-tier components into a cohesive computing environment. And as object-oriented programming has entered mainstream application development, CORBA has emerged as the standard middleware solution for integrating distributed e-business components that are implemented in disparate languages – Java, C++ and others.

Distributed applications require a completely different level of monitoring and diagnostic support than traditional single-process applications. While the factors causing unexpected behavior and failures in a single process might be simple and easy to anticipate, a distributed system can suffer from any one or more of a whole range of bugs. Let's briefly review seven of the most common problems:

- **Performance bottlenecks** can appear in a distributed application when a complex operation is performed at a time-critical point, and can substantially slow down your application's overall performance.
- **Network resource limitations** can cause a distributed system to fail when the size of the system is ramped up. Scalability problems may not occur within your test configurations, but can appear later during deployment in the form of limited connections or insufficient bandwidth.
- **Network failures** can often partially

afflict a complex network. As the application developer, it behooves you to detect and circumvent each point of failure.

- **Race conditions** can occur if parallel working modules of a distributed application aren't properly synchronized to prevent different modules from producing contradictory results. Synchronization errors are difficult to detect because they tend to be sporadic and aren't easily reproduced.
- **Deadlocks** can appear when the synchronization protocol between modules prevents each from completing its task. Like a race condition, a deadlock often appears only in a special situation and can be difficult to locate.
- **Design errors in control flow** can occur very easily. The control flow in a distributed application is usually much more complex than in a single-process application, leading to a wide variety of design errors. Unlikely events such as exceptions and failures within multiple modules can be especially difficult to handle.
- **Timeout failures** can occur owing to delays and bottlenecks in the network that cause distributed parts of an application to time out and produce a failure. Such a failure may propagate through the rest of the application if you don't handle it properly.

Over and above these various problems, in a distributed system diagnosis is more complicated than debugging a conventional single-process application. To set up and step through a test case can be very time-consuming when

using code debuggers for the distributed modules of a system. And correlating message entry and exit points among numerous processes, each with its own code debugger, can quickly become impractical. Since you're not testing the application in real execution time, time-critical failures such as bottlenecks, race conditions, deadlocks and timeouts can't be detected. Conventional debugging rarely detects scalability problems either.

Monitoring Messages Between CORBA Objects

One good and effective way of diagnosing distributed systems is through monitoring communication between the various distributed components. The objective of this article is to demystify the CORBA communication bus by showing you how to capture the details of messages passed between CORBA objects. Such monitoring lets you observe and record method invocations and exceptions selectively, helping you avoid or eliminate bottlenecks, race conditions and other potential failures that might otherwise impede the performance of your application.

Let's look at what goals you should bear in mind as you monitor these messages.

- **Distributed debugging:** You need to monitor them during the development and test phases of a project. That way, you'll uncover problems before an application is deployed. When the application goes live, communication details should be logged to enable performance analysis and

Object Management Group

www.omg.org

to make it possible to troubleshoot unexpected failures quickly. You should be able to activate monitoring on the fly without having to stop and restart the application.

- **Application-level communication details:** You should observe request-reply details as they occur at the application level. For example, “The buy method of the stock_exchange object was called using a stock symbol of SEGU and a share amount of 1000.” Monitoring at the application level requires an understanding of all CORBA data types and of complex user-defined types. Details captured about each message should include request ID, interface name of the target object, method being invoked, parameter values, timing data, process IDs, host IDs and any thrown exceptions.
- **Dynamic activation:** Make sure that application objects are completely unaware of any active monitoring. You should be able to dynamically turn monitoring on or off and specify which communication details to observe while your application is running.
- **Filter criteria:** Ensure that it's easy to filter traffic and thus monitor only those interfaces, methods and parameters that you're interested in. Make sure too that it's possible to stipulate how many times a particular method will be observed and at which communication entry and exit points.
- **Timing analysis:** Use message timestamps and timing data to help identify server latency, message travel time and client wait time – information that's extremely helpful for diagnosing and resolving timing-related problems.
- **Data recording:** Record monitored communication to enable logging message activity or analyzing results. It should be possible to parse and sort the recorded data.

How CORBA Communication Works

CORBA can be conceptualized as a communication bus for distributed

objects. In a CORBA system the “client/server” terminology applies within the context of a specific request. In other words, if object A invokes a method on object B, A is the client and B is the server; if B then calls A, the roles are reversed.

The Object Request Broker (ORB) is the mediator, responsible for brokering interactions between objects. Its job is to provide object location and access transparency by enabling client invocations of methods on server objects (see Figure 1). If the server interface is known at build time, a client can connect – or bind – to a server object statically. If unknown, it can use dynamic binding to ascertain a server's interface and construct a call to the corresponding object.

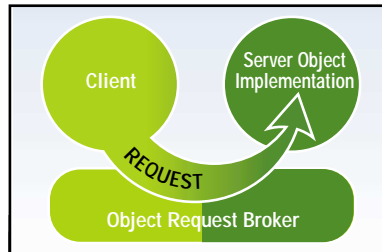


FIGURE 1 A client invokes an object's methods through the ORB.

Exported server interfaces are specified in the CORBA standard interface definition language (IDL). You don't write server implementations in IDL: an interface description is mapped instead, using an IDL compiler, to native language bindings such as Java or C++. This allows each programmer to write source code independently in whichever language may be the most appropriate. A Java program, for example, can access a server object implemented in C++ – the Java programmer merely invokes methods on the server as though they're local Java method calls. Figures 2 and 3 illustrate, respectively, an IDL description for a CORBA server and a Java client that calls a corresponding object implementation.

In Figure 2 Account is an interface that corresponds to a class implemented in a server. IDL attributes define the properties of a class (e.g., balance). The

IDL compiler maps attributes to “get” and possibly “set” methods. Operations define the methods to be implemented by the server (e.g., make_deposit and make_withdrawal). Their parameters must be explicitly identified in the interface description as in, out or inout. Many other features are supported by IDL, such as inheritance for specifying derived interfaces, modules for establishing naming scopes and exceptions that are supported by an interface or raised by operations.

The IDL compiler generates a skeleton that's linked to the server program and provides static interfaces to call methods of an object implementation. The skeleton unmarshals methods and parameters that come from a client via the ORB. The IDL compiler also generates a client stub that's linked to programs that will statically invoke server methods through the associated interface. The client stub maps a CORBA server object to a native object in the client's language (see Figure 3). The stub acts as a proxy for remote server objects by marshaling methods and parameters to be transmitted via the ORB. CORBA also supplies the dynamic invocation interface (DII) for client programs to discover server interfaces and construct method calls at runtime. The DII requires the use of the CORBA interface repository, which contains compiled IDL descriptions that can be interrogated programmatically (see Figure 4).

The CORBA standard guarantees interoperability between applications built using different vendors' ORBs. The Internet InterORB Protocol (IIOP) defines standard message formats, a common data representation for mapping IDL data types to flat messages and a format for an interoperable object reference (IOR) over TCP/IP networks. In other words, IIOP is the CORBA wire-level protocol.

CORBA communication typically consists of a request message and a reply message. Most ORBs implement interceptors that permit these IIOP messages to be traced at the four points shown in Figure 5: SendRequest, ReceiveRequest, SendReply and ReceiveReply.

```
// IDL
interface Account
{
    //Attributes
    readonly attribute float balance;

    //Operations
    float make_deposit( in float amount );
    float make_withdrawal( in float amount );
};
```

FIGURE 2 Simple interface description specified in IDL

```
// Java
//Deposit $100.25 into "accountRef"
try {
    //...
    float result = accountRef.make_deposit((float)100.25);
} catch (org.omg.CORBA.SystemException ex) {
    System.out.println("EXCEPTION: " + ex);
}
```

FIGURE 3 Excerpt from a Java client

n-ary

www.n-ary.com

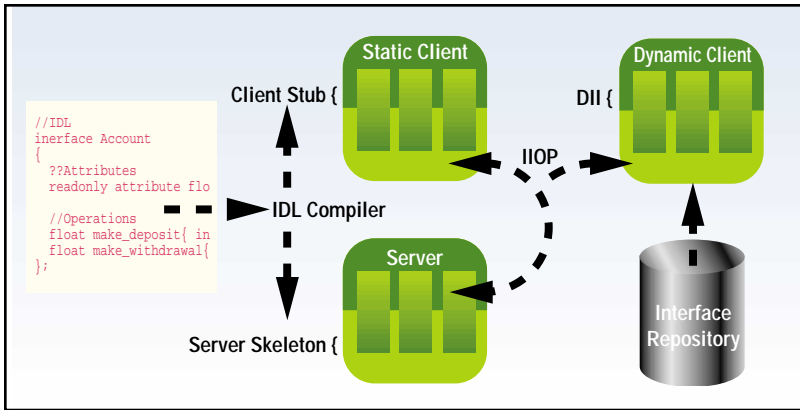


FIGURE 4 An IDL description is mapped to a server skeleton and client stub.

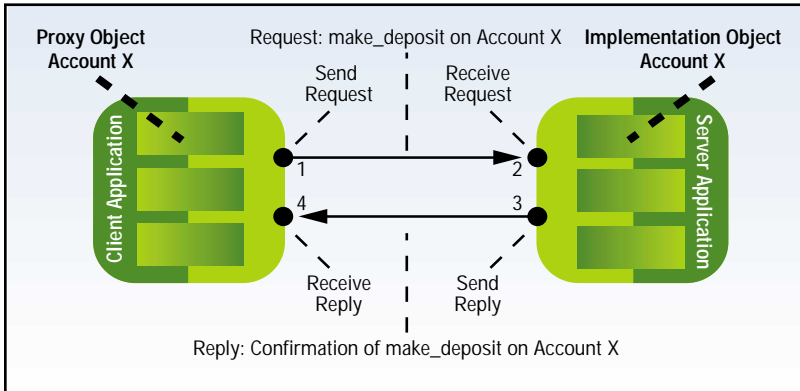


FIGURE 5 Request and reply messages can be intercepted at four points.

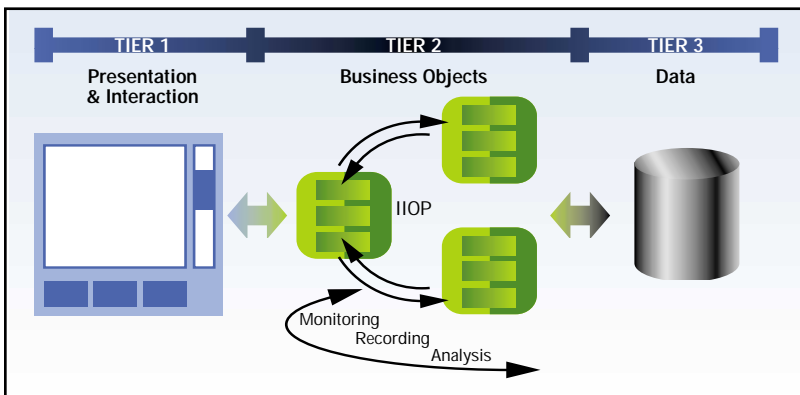


FIGURE 6 Intercepting and interpreting IIOP message traffic allows you to monitor, record and analyze business object communication.

An Architecture for Monitoring Communication

Now that we've discussed the goals of monitoring messages between distributed components and reviewed how CORBA communication works, let's look at an architecture for monitoring in a CORBA environment.

Intercepting and interpreting IIOP messages can be achieved using four types of architectural components: Probe, Profile, Collector and Observer. Each monitored CORBA process – whether acting as a client, server or both – contains a Probe object that captures messages based on the filter criteria as

specified by an active Profile. A Profile can be created, updated or uploaded to a Probe at any time. The intercepted data is recorded by the Probe, read by a Collector and transmitted to an Observer. The Observer is the primary collection point for aggregating data from multiple Collectors, and the data it records can then be viewed and analyzed (see Figure 6).

Given the absence of standardization in the area of CORBA monitoring and diagnostics, the design and implementation of a monitoring architecture will vary depending upon who's creating it – the application developer, the ORB vendor or (preferably) an independent tool vendor. The issue of standardization will

be discussed later. First, let's explore each of our architectural components in detail.

PROBE

Most ORBs provide interceptors that allow the creation of a Probe object, which captures and records IIOP messages, within each monitored CORBA process. Only one Probe object is necessary per process, regardless of the number of business objects created. The business objects are completely unaware of the Probe, which means the application's business logic doesn't take the Probe into account, except for the code that creates an instance of the Probe object. (Such instrumentation code would typically be placed in the main routine after initializing the ORB – outside the actual business objects.)

PROFILE

A Profile specifies filter criteria used by a Probe while collecting messages. It's a dynamically configurable filter that can be uploaded to a Probe residing within a running program (see Figure 7) and it serves three purposes:

1. It scopes the traffic being observed to include only the interfaces, methods and parameters of interest.
2. It indicates how many times a particular method should be observed.
3. It specifies any or all of the four possible communication points to capture data, i.e., SendRequest, ReceiveRequest, SendReply and ReceiveReply.

A Profile might specify, for example, "Monitor up to 20 invocations of the method `make_deposit` at the ReceiveRequest and SendReply points." The CORBA interface repository is useful for a Profile editor or similar tool to determine details about available object interfaces. This allows you to create or modify a Profile, which you can then upload to a Probe within a running process.

COLLECTOR

A collector serves as the registration point in the monitoring architecture for

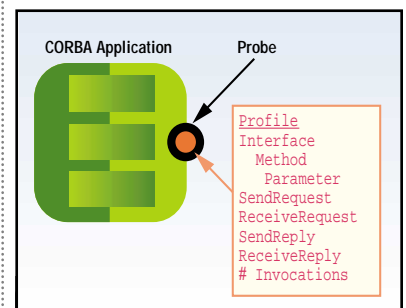


FIGURE 7 Instrumented CORBA application

ADVERTISER INDEX

ADVERTISER	URL	PH	PG	ADVERTISER	URL	PH	PG
4TH PASS	WWW.4THPASS.COM	877.484.7277	87	KL GROUP INC	WWW.KLGROUP.COM/TICKET	888.361.3264	21
ACTIVATED INTELLIGENCE	WWW.HEADLINEWATCH.COM	212.896.8220	127	KL GROUP INC	WWW.KLGROUP.COM/GREAT	888.361.3264	101
AJILE SYSTEMS	WWW.AJILE.COM	408.557.0829	134	KL GROUP INC	WWW.KLGROUP.COM/COLLECT	888.361.3264	212
ALLAIRE CORPORATION	WWW.ALLAIRE.COM/DOWNLOAD	888.939.2545	25	METAMATA INC	WWW.METAMATA.COM	510.796.0915	89
AMAZON.COM	WWW.AMAZON.COM		121	MICROSOFT	MSDN.MICROSOFT.COM/TRAINING		11
APPEAL VIRTUAL MACHINES	WWW.JROCKIT.COM	46 8402 28 73	27	THE MIDDLEWARE COMPANY	WWW.MIDDLEWARE-COMPANY.COM/INFO		127
BLUESTONE SOFTWARE	WWW.BLUESTONE.COM	888.BLUESTONE	4	MODIS SOLUTIONS	WWW.IDEA.COM	703.821.8809	69
BUZZEO	WWW.BUZZEO.COM	800.804.4724	63	N-ARY	WWW.N-ARY.NET	877.849.6833	135
CAPE CLEAR	WWW.CAPECLEAR.COM	353.1.618.2071	47	NO MAGIC	WWW.MAGICDRAW.COM	303.914.8074	5
CAREER OPPORTUNITIES		800.846.7591	177-209	NORTHWOODS SOFTWARE CORP.	WWW.NWOODS.COM	800.226.4662	159
CEREBELLUM SOFTWARE	WWW.CEREBELLUMSOFT.COM	888.862.9898	33	OBJECT MANAGEMENT GROUP	WWW.OMG.ORG	781.444.0404	139
CERTICOM	WWW.CERTICOM.COM	800.476.0196	111	OOPSLA 2000	WWW.OOPSLA.ACM.ORG	503.252.5709	103
CIMMETRY SYSTEMS, INC.	WWW.CIMMETRY.COM	800.361.1904	152	PERSISTENCE	WWW.PERSISTENCE.COM	650.372.3600	71
CODEMARKET	WWW.CODEMARKET.NET/SPECIALOFFER	914.638.2159	117	POINTBASE	WWW.POINTBASE.COM/JDJ	877.238.8798	93
COMPUTERWORK.COM	WWW.COMPUTERWORK.COM	800.691.8413	134	PROGRESS SOFTWARE	WWW.SONICMQ.COM/AD1.HTM	800.989.3773	2
COMPUWARE	WWW.COMPUWARE.COM/NUMEGA	800.4.NUMEGA	29	PROSYST	WWW.PROSYST.COM	678.366.5075	95
CRUEL WORLD	WWW.CRUELWORLD.COM	650.847.3581	81	PROTOVIEW	WWW.PROTOVIEW.COM	650.847.3588	3
DEVELOPMENTOR	WWW.DEVELOPMENTOR.COM	503.681.4724	152	QUICKSTREAM SOFTWARE	WWW.QUICKSTREAM.COM	888.769.9898	154
ELIXIR TECHNOLOGY	WWW.ELIXIRTECH.COM/DOWNLOAD	65.532.4300	73	SEGUE SOFTWARE	WWW.SEGUE.COM	800.287.1329	55
ELIXIR TECHNOLOGY	WWW.ELIXIRTECH.COM/ELIXIRREPORT	65.532.4300	123	SIC CORPORATION	WWW.SIC21.COM	822.227.398801	97
ELIXIR TECHNOLOGY	WWW.ELIXIRTECH.COM/DOWNLOAD	65.532.4300	129	SILVERSTREAM	WWW.SILVERSTREAM.COM	978.262.3000	211
EMBARCADERO TECHNOLOGIES	WWW.EMBARCADERO.COM/DESIGN	415.834.3131	151	SIMPLEX KNOWLEDGE COMPANY	WWW.SKC.COM	914.620.3700	133
EMBARCADERO TECHNOLOGIES	WWW.EMBARCADERO.COM/ADMINISTER	415.834.3131	153	SLANGSOFT	WWW.SLANGSOFT.COM	972.2.648.2424	136-137
EMBARCADERO TECHNOLOGIES	WWW.EMBARCADERO.COM/DEVELOP	415.834.3131	155	SOFTWARE AG	WWW.SOFTWAREAG.COM/BOLERO	925.472.4900	75
EVERGREEN	WWW.EVERGREEN.COM/JDJ.HTML	408.926.4500	91	SOFTWIRED	WWW.SOFTWIRED-INC.COM/IBUS	411.445.2370	99
FIORANO	WWW.FIORANO.COM	408.354.3210	109	STARBASE	WWW.STARBASE.COM	888.STAR.700	105
FLASHLINE	WWW.FLASHLINE.COM	800.259.1961	67	STERLING SOFTWARE	WWW.COOLJOECHALLENGE.COM	972.801.6000	106-107
GEMSTONE	WWW.GEMSTONE.COM/WELCOME	503.533.3000	31	SYBASE	WWW.SYBASE.COM/PRODUCTS/EASERVER	800.8.SYBASE	51
GENERIC LOGIC, INC	WWW.GENLOGIC.COM	413.253.7491	94	SYNTION AG	WWW.SYNTION.COM	49.89.52.3045.0	145
HIT SOFTWARE	WWW.HIT.COM	408.345.4001	49	SYS-CON RADIO	WWW.SYS-CON.COM	888.886.8769	124
HOTDISPATCH.COM	WWW.HOTDISPATCH.COM	650.234.9752	57	THINWEB TECHNOLOGIES	WWW.THINWEB.COM	877.THINWEB	23
IAM CONSULTING	WWW.IAMX.COM	212.580.2700	83	TIDESTONE TECHNOLOGIES	WWW.TIDESTONE.COM	800.884.8665	65
INETSOFT TECHNOLOGY CORP	WWW.INETSOFTCORP.COM	732.235.0137	45	TOGETHERSOFT CORPORATION	WWW.TOGETHERSOFT.COM	919.833.5550	6
INETSOFT TECHNOLOGY CORP	WWW.INETSOFTCORP.COM	732.235.0137	56	UNIFY CORPORATION	WWW.EWAVECOMMERCE.COM	800.GOUNIFY	85
INFORMATION ARCHITECTS	WWW.IA.COM	704.365.2324	37	UNIFY CORPORATION	WWW.SERVLETEXEC.COM	678 366.3211	77
INFORMATION ARCHITECTS	WWW.IA.COM	704.365.2324	39	VERGE TECHNOLOGIES GROUP INC	WWW.EJIP.COM		119
INFORMATION ARCHITECTS	WWW.IA.COM	704.365.2324	41	VISICOMP	WWW.VISICOMP.COM/JDJ6	831.335.1820	35
INFORMATION ARCHITECTS	WWW.IA.COM	704.365.2324	43	VISUALIZE INC.	WWW.VISUALIZEINC.COM	602.861.0999	159
INFORMIX/ CLOUDSCAPE	WWW.CLOUDSCAPE.COM	510.239.1900	18-19	VSI	WWW.BREEZEXML.COM	800.556.4874	79
INSIGNIA SOLUTIONS	WWW.INSIGNIA.COM/JEODE	800.848.7677	15	WEBGAIN	WWW.WEBGAIN.COM	888.822.3409	9
INTUITIVE SYSTEMS, INC	WWW.OPTIMIZEIT.COM	408.245.8540	59	WEBGAIN	WWW.WEBGAIN.COM	888.822.3409	53
IONA TECHNOLOGIES	WWW.IONA.COM/JPAS.HTM	781.902.8281	115	WEBGAIN	WWW.WEBGAIN.COM	888.822.3409	61
JAVA DEVELOPER'S JOURNAL	WWW.JAVADEVELOPERSJOURNAL.COM	914.735.0300	38,145	WEBVISION	WWW.WEBVISION.COM	800.531.5057	13
JAVACON2000	WWW.JAVACON2000.COM		146-147	WINTERTREE SOFTWARE	WWW.WINTERTREE-SOFTWARE.COM	800.340.8803	158
JAVACON2000	WWW.JAVACON2000.COM		149	XML DEVCON 2000	WWW.XMLDEVCON2000.COM		161-176
JAVAONE CONFERENCE	HTTP://JAVA.SUN.COM/JAVAONE	888.886.8769	131	YOUCENTRIC	WWW.YOUCENTRIC.COM/NOBRAINER	888.462.6703	113
JDJ STORE.COM	WWW.JDJSTORE.COM	888.303.JAVA	156-157	ZUCOTTO	WWW.ZUCOTTO.COM	613.789.0090	125

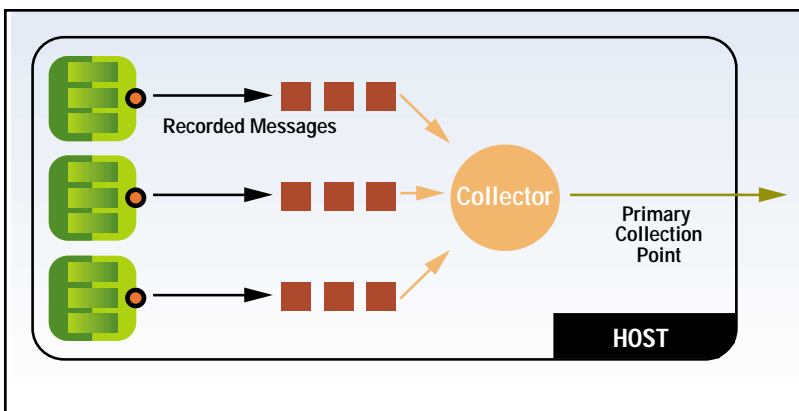


FIGURE 8 Collecting intercepted data

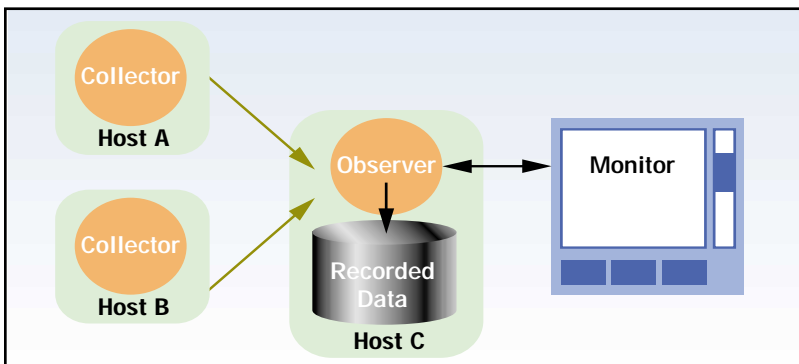


FIGURE 9 Recording and monitoring

the local application programs. A Probe writes intercepted IIOP messages for subsequent retrieval by the Collector using the fastest possible mechanism and format so that the Probe doesn't become a bottleneck by blocking the message traffic. As the data is written, the Collector reads the messages and transmits them to a primary collection point (see Figure 8). A Collector may also transmit additional relevant data about monitored processes to the primary collection point.

OBSERVER

The Observer is the primary registration and collection point for all Collec-

tors across the distributed environment. As data is transmitted to the Observer, it's written to a global database viewable in real time to permit the analysis of system performance (see Figure 9).

The information about each message captured includes:

- Sequence number of the request
- Name of the interface containing the operation or attribute
- Name of the operation or attribute in the request
- Amount of time the server took to process the request
- Amount of time spent by the request on the wire

- Total time between the request being sent and reply being received (server time + travel time)
- Parameters in the request at each of the communication points
- System process IDs and names of the hosts for the server and the caller
- Timestamps at each of the communication points
- Details about any thrown exception including the communication point where the exception was raised

Figure 10 illustrates a simple captured message.

Summary: What You Can Do

Gaining insight into distributed system behavior can be complicated, but by performing application-level monitoring during a project's development and test phases you can uncover problems prior to deployment and thereby ensure that your application is reliable. Then, when it subsequently goes live, you can capture communication details to allow performance analysis and quick troubleshooting of unexpected failures. Monitoring and diagnostics in your CORBA system can be achieved using a commercially available tool such as Segue Software's SilkObserver or by building custom instrumentation into your application. Either way, differing techniques may be applied depending on which ORBs you're using and what the specific monitoring objectives are for your system.

The OMG's Test Special Interest Group is in the process of standardizing distributed instrumentation and control for CORBA systems. (A Request for Proposals is being drafted as of this writing and will be issued this summer.) The group is defining a common set of instrumentation capabilities for use in the management, debugging and profiling of multivendor CORBA-based systems. These capabilities will likely rely on the OMG's anticipated standard for portable interceptors. The Test SIG's efforts should yield an interface specification for controlling object execution, returning state information and providing other useful functions that today are performed inconsistently – if at all – across CORBA implementations. When complete, the OMG's efforts in this area should result in consistent mechanisms for distributed monitoring and diagnostics while using the ORB of any vendor. You can check out the progress of the Test SIG on the OMG Web site (www.omg.org) or by sending an e-mail to info@omg.org.

AUTHOR BIO
 Todd Scallan is the vice president in charge of Segue Software's California Development Laboratory. He holds a BS in electrical engineering from Lehigh University and an MS in computer engineering from Syracuse University.

PROCESS INFORMATION		TIMING INFORMATION	
Interface: Account Method: make_deposit		Total Time: 32 ms Travel Time: 22 ms Server Time: 10 ms	
<u>Client</u>	<u>Server</u>		
Process: BankClient Host: cypress.segue.com Profile: client-filter Process ID: 171235899	Process: BankServer Host: redwood.segue.com Profile: server-filter Process ID: 171063782		
SENDREQUEST _ >		> _ RECEIVERREQUEST	
Time Stamp: Fri Mar 24 07:52:57:995 PST 2000 Parameters: in float amount = 100.25 Return: Void		Time Stamp: Fri Mar 24 07:52:58:015 PST 2000 Parameters: in float amount = 100.25 Return: Void	
RECEIVEREPLY _ <		< _ SENDREPLY	
Time Stamp: Fri Mar 24 07:52:58:027 PST 2000 Parameters: Return: (float) 879.67		Time Stamp: Fri Mar 24 07:52:58:025 PST 2000 Parameters: Return: (float) 879.67	

FIGURE 10 Details of a captured message

tscallan@segue.com

Syntion AG

www.syntion.com

Java Developer's Journal

www.javadevelopersjournal.com

JavaCon 2000

www.javacon2000.com

JavaCon 2000

www.javacon2000.com

Extending Your Applications with Bean Scripting Framework

BSF provides a universal
scripting platform for Java

WRITTEN BY RICK HIGHTOWER

Part 4 of a series discussing the many languages that compile and/or run on the Java platform

Do you remember the operating system religious wars? Mac OS versus Windows, Windows NT versus UNIX, OS/2 versus Windows NT. Or how about the text editor wars – VI versus Emacs? It may seem silly for programmers to become involved so passionately with the technology they work with, but if you spend more time with your VI text editor than with your family, I guess you do get kind of attached. I'm certain, for example, that we all have our favorite programming language...

Recently, for example, I was explaining to a colleague why I like to prototype things in JPython. He stared at me blankly and asked what at first seemed an innocent enough question: "Why?"

I spent the next five minutes explaining – I didn't even take a breath. To which he again responded "why?" before going on to expound on how Java was the perfect language and how you don't need another one. At this point I found it better not to continue the conversation. I try to avoid religious attachment to technology. (See the March issue of *JDJ* [Vol. 5, issue 3] for more about JPython.)

I've had similar experiences when talking to VB, Perl, Delphi, C++ and REXX programmers. Perl programmers seem particularly attached to their language. Developers have favorite languages just like developers have their favorite editor. (Mine is Emacs.)

Java is one of my favorite programming languages. However, it's more than just a language, it's a platform, and Java the platform runs many, many programming languages. (See the first article in this series to learn more about programming languages that run in the Java Virtual Machine ["Programming Languages for the JVM," Vol. 5, issue 2].)

Now let's say that you wanted to make available a set of services from a Java-based application via a scripting language, as LotusScript does for Notes or VBA (Visual Basic for Applications) does for Excel. Which scripting language do you use? It's more or less impossible to pick one without leaving someone bent out of shape. Remember, everyone has their favorite. You want your application services to be inclusive, not exclusive.

What if you could support *all* of the major scripting languages with the same or less effort than it took you to support *one*? Essentially, with Bean Scripting Framework (BSF) from IBM, you can support Perl, Python, NetRexx, JavaScript and even VBScript. Cool beans!

What's more, BSF is going to add standard ways to debug scripts, a major flaw in some scripting languages. (I know, I know: real programmers don't use visual debuggers. But admit it, you do sometimes – and so do I, but don't tell anyone.) BSF brings standard support for many programming languages to the Java platform.

Why Scripting Languages?

The BSF white paper states the following: "As component-oriented software development becomes more and more commonplace, scripting is fast becoming a key development methodology...Scripting is a natural counterpart to component oriented development – components can be written in standard object-oriented languages and then 'glued together' to

form applications using scripting languages....[Scripting languages] are a natural fit in the component oriented development world." (Quote taken from *Bean Scripting Framework: A Scripting Architecture for the Java Platform* written by Rick Rineholt, Sam Ruby, Matthew J. Duftler and Sanjiva Weerawarana.)

I believe the above echoes my same sentiments from the first article in this series. Scripting languages and components go together like a horse and carriage: having a standard way for scripts to talk to Java components opens up a lot of possibilities. BSF endeavors to provide a standard for scripts to communicate with Java components. (Please refer to the first article for more information on components and scripting languages.)

What Is BSF?

An application that uses BSF can use scripting – and become scriptable – using any BSF-supported language. Thus, when BSF adds support for additional languages, your application will automatically support the new languages.

The official Java platform from Sun doesn't have a standard scripting architecture, but IBM is submitting BSF to JavaSoft as a Java Specification Request, so BSF may become the basis of the Java platform standard extension for scripting. Even if it doesn't, BSF is likely to be the de facto standard for scripting integration for the Java platform.

Why BSF?

The advantages of having a standard scripting architecture like BSF is as follows:

- It replaces the ad hoc approach to script integration.
- It enables applications to support a lot of scripting languages.
- Its scripts enable "nonprogrammers" to extend your application.
- Services like debugging can be shared.

What Languages Does BSF Support?

Currently, the BSF supports the following pure Java platform languages:

- Netscape's Rhino (JavaScript)
- Jacl (TCL)
- JPython (Python)
- NetRexx (REXX variant)
- Bean Markup Language (developed by IBM)
- LotusXSL
- Pnuts

The scripting language doesn't have to be implemented in Java to be supported by BSF. For example, IBM adds support for Perl and VBScript. Essentially, all active scripting languages including VBScript and JScript are supported via BSF's support for Microsoft Active Scripting Framework (MASF).

Java- Con 2000

www.java-

con2000.com

BSF is the Java version of MASF, which is weird because some languages that are supported by MASF are also supported by BSF. Thus you can mix classic Python with JPython. This is good for legacy integration. In addition, every language that gets added to MASF automatically gets added to BSF. Pretty tricky, huh?

BSF is to MASF as JDBC is to ODBC. Just as you can have pure database drivers in JDBC or use native ODBC drivers, you can have pure Java scripting languages or use native scripting languages in BSF.

Think of all the people in the world who have done VBScript (or VBA). Now you can allow them to participate in your development efforts. Not only that, but you have another integration point with COM via MASF. With MASF, you can instantiate COM objects. (Don't worry, you 100% Pure zealots, you can still use BSF scripting in a Pure environment as well.)

Distribution

BSF is currently freely available – source and all. BSF will be developed under an open-source model in the very near future. If you have a favorite scripting language that's not supported by BSF, you should consider defining your own BSFEngine, which can be plugged into the framework.

Architecture

The BSF architecture consists primarily of two components: BSFManager and BSFEngine. The BSFManager is a common interface to scripting languages: you use it to access scripts from your applications. The BSFEngine interface provides a common interface for BSF to interact with a scripting language. The JDBC is to a JDBC Driver as BSF is to a BSFEngine. Essentially, a BSFEngine is a scripting language driver.

Getting Started

I downloaded the latest version of BSF and tried it out with one of my favorite scripting languages: JPython.

The first step on every journey like this is usually a download or two and this adventure is no different. I went to IBM's alphaWorks Web site, looked up BSF and downloaded the bsf21.zip before extracting it to C:\BSF-2.1.

The zip file contains all the files you need to get started: API documents, a getting started guide, source code and library files (JAR files). The ReadMe file has most of the information you need to get started. The getting started guide introduces you to most of the concepts behind BSF with some code example snippets.

Even with all of the above, setting up BSF is not for the averagely motivated person. The BSF guide is good but it needs a little more meat – it's not a step-by-step tutorial. If you like to tinker, then BSF is for you, but if you rarely wander from the confines of your favorite Java IDE, BSF may not be for you at this stage.

Eventually I got some sample code that I'd written to work. I had a problem and needed to look at the BSF source to figure out what was happening. It was a little tougher than I thought it would be (see the sidebar for details), but it wasn't impossible.

One of the keys to getting started smoothly is to make sure the JAR files for BSF and those for your scripting language are on the CLASS-

HAVING THE SOURCE IS KING

The cool thing about having the source is that you can peek at the man behind the curtain and see what's really going on. The problem was just a configuration issue; however, the BSF code caught a very meaningful exception and then threw a very vague exception. Once I saw what exception it was catching and then throwing (by looking at the source code for BSF), I could figure out what was really happening: I had misspelled one of the JAR files and consequently the BSFManager couldn't find the BSFEngine for JPython.

PATH. Two JAR files ship with BSF that you need on your classpath: bsf.jar and bsfengines.jar. The bsf.jar has the core BSF files. The bsfengines.jar has the language drivers, i.e., the language engines. You also need the JAR files for your language on the classpath, e.g., if you are using JPython you need JPython.jar on your classpath. (JPython.jar ships with the JPython distribution.) For NetRexx you need NetRexxC.jar, NetRexxR.jar and tools.jar on your classpath; for TCL (JACL) you need JACL.jar; and for JavaScript you need js.jar and jstools.jar.

The example I'm going to highlight in this article will focus on integrating JPython. However, with slight modification you can incorporate TCL, JavaScript, NetRexx and so on. I'll leave the modification up to you.

For the code example, we'll map in an instance of a class to the BSFManager, load a script and then execute it. The script will have code that interacts with the class instance we mapped in the BSFManager. The class we'll use should be familiar to you if you've been following this series. Basically, it's a variation of the Statistics class from the last two articles.

The Stats class (see Listing 1) is a simple class that figures out the mean, mode and median for a given set of numbers. The script that we're going to use to manipulate this instance of Stats is very short (see Listing 2) – it prints out the mean, mode and median price of a list of houses.

```
print "The mode of the houses is %2.2f" % houses.mode
print "The mean of the houses is %2.2f" % houses.mean
print "The median of the houses is %2.2f" % houses.median
```

The houses variable is an instance of the Stats class that gets mapped into the BSFManager.

The steps to use a scripting language in BSF are as follows:

1. Register the language with the BSF manager.
2. Load the scripting engine.
3. Map in your application objects that you want the script to have access to.
4. Load the script and execute it.

In addition to loading and executing scripts, you can execute arbitrary expression of your scripting language. Listing 3 shows the code that does all of the above steps as well as executes a JPython expression.

The first thing that Listing 3 does is to register JPython with the BSF manager:

```
BSFManager manager = new BSFManager ();
//Register JPython into the scripting manager.
String[] extensions = {"py"};
manager.registerScriptingEngine ("jpython",
    "com.ibm.bsf.engines.jpython.JPythonEngine",
    extensions);
```

The manager has a method called registerScriptingEngine that takes three arguments: the name of the scripting language; the fully qualified class name of the BSF engine corresponding to the scripting language; and an array of string that corresponds to all the possible file extensions for the scripting language.

Remember, the BSF ships with several engines for various languages, so you can easily change the above to work with JavaScript or NetRexx or whatever. Once you've registered the scripting language, you can load its engine and start working with scripts. To load the engine, call the manager's loadScriptingEngine method:

```
//Load JPython engine
BSFEngine jpythonEngine = manager.loadScriptingEngine ("jpython");
```

After the scripting engine is loaded, you can start evaluating expressions and executing scripts. A scripting language expression is evaluated by calling the eval method on the manager.

Interview...with

SAM RUBY & SANJIVA WEERAWARANA

Two key members of the BSF development team, Sam Ruby and Sanjiva Weerawarana, helped me considerably in developing and researching this article. I had a chance to open up a dialog with them at the time about the future of BSF and as you'll see from the following account of our conversation, they were extremely helpful.

RICK HIGHTOWER: *Is the Apache Tomcat integration committed yet? [This provides the ability for JSP in Tomcat to work with other scripting languages like JPython, TCL, NetRexx, JavaScript, Prnuts and so on.] If not, what's the ETA?*

Sam Ruby: Unfortunately, no. Since it extends the spec and Tomcat is meant to be a reference implementation, I decided to make it available first as a JSP taglib. The Jakarta Project management committee decided to make a separate jakarta-taglib tree, so I opted to wait for that. And now there have been delays in making that project available. Sigh. I expect this to have been cleared up by April. I did integrate it with the build tool used by most Java Apache projects (ANT). I expect by May to have moved on to the XML-Apache projects – in particular, I plan to integrate it [BSF] with the XSP component of XML-cocoon. XSP is an XML-centric implementation roughly analogous to JSP or ASP. Fortunately, it doesn't have a standards process to work through! I don't know how to adequately express in a sound bite how standards are simultaneously the most important yet frustrating things to deal with as a developer.

RH: *How soon will you go open source with BSF? The last I heard was that you have legal, management and executive approval and that you had only one checkpoint left.*

SR: One last issue to resolve and everything is a go (one lawyer decided to ask another lawyer to check into a patent issue). Should be a matter of days.

RH: *What's the current progress regarding adding debugging support?*

SR: This is only just now getting staffed.

RH: *What is the current progress of creating components (JavaBeans) with any BSF scripting language?*

Sanjiva Weerawarana: We're pretty much done with it and expect to release it to alphaWorks in the next two weeks. There are a few constraints of the current implementation but the basic idea is proven with what we've done.

RH: *What is the total number of languages supported?*

SW: All MS ActiveScript languages (including VBScript

and JScript), BML, JACL, JavaScript (Rhino), JPython, NetRexx, TCL and XSLT. Prnuts support for BSF is available with that language. Support for LotusScript and Perl are under active development. You count them. [13+]

RH: *What kind of resources do you have dedicated to BSF?*

SW: None of your business! <grin> Seriously, I doubt that the number has ever been much more than four inside IBM. Once it's open source, I expect the number to grow significantly.

RH: *Are there any other future directions that are not covered in the white paper that you sent?*

SW: The biggest one I have in mind is that the BSF concept could be implemented for C as well. Currently Win32 has a scripting architecture – guys who develop in UNIX-land, however, don't have anything similar. It's quite doable to take BSF and do a BSF-for-C type of thing where the infrastructure assumes a C runtime rather than a Java runtime. The thing that you really need is something that pretends to [be] the equivalent of reflection; this isn't that hard. I think this could be a very attractive piece of software! We also plan on working on the compilation side of BSF quite a lot. One of our guys has developed a killer way to generate code and we're merging it with some other stuff and hope to dramatically improve the flexibility of the "compileScript" method of BSF. Yes, the description in this paragraph is intentionally vague!! I think the issues with debugging and better error handling were listed in the paper I sent you.

RH: *Any updates?*

SW: Hmm. I think the user's guide with the latest version on alphaWorks is pretty up to date. We haven't been working on BSF much because we've been waiting for the open-source clearance to go [through].

RH: *Where can I find more information about BSF?*

SR: The primary URL is www.alphaworks.ibm.com/aw.nsf/techmain/bsf.

RH: *Can you provide a list of known products that are using BSF? (I heard NetBeans was going to add support for BSF to their IDE.)*

SR: WebSphere and ANT are the ones that I've been involved with, as well as several unannounced IBM products.

SW: Please add Apache Xalan to this list – it uses BSF for implementing the XSLT extensions.

RH: *What is the status of submitting BSF to JavaSoft as a Java Specification Request as an extension for scripting?*

SR: I've seen the draft submissions, so I know this is in progress. Sanjiva will have the up-to-the-minute status on this.

SW: The status on this is that I've finished filling in the JCP template as well as our internal stuff and it's with the IBM approval team awaiting evaluation. Those guys meet like once a month or so...so it'll get taken care of at the next meeting. I expect easy passing because we had preliminary approval from before. ☺

Embar
cadero
p/u
www.
embarcadero.
com

```
//Execute an expression.  
Object result = manager.eval ("jpython",  
"testString", 0, 0, "2+32");  
System.out.println("eval="+result);
```

The eval method takes five arguments as follows: the name of the scripting language, the name of the file name associated with the expression, the row of the expression, the column of the expression, and – last – the expression. The above expression is "2+32". When the above code runs, it prints out 34.

Of course, in addition to evaluating expressions, you can execute scripts. Before evaluating scripts you can map objects (or an entire object model) into the BSF manager. All objects that are mapped in the BSF manager are available to the scripts.

The following code instantiates an instance of Stats, and then maps that instance into the BSF manager:

```
//Map some objects that the script will  
use.  
double[] houses=new double [] {100.0e3,  
130.0e3, 140e3, 150e3};  
Stats stats = new Stats(houses);  
manager.declareBean("houses", stats,  
Stats.class);
```

One way to map objects into the BSF manager is to call the declareBean method. The declareBean method takes three arguments:



the script variable name of the bean (object), the instance and the class of the instance.

After you've mapped in your application's scriptable objects into the manager, you can load and run a script as follows:

```
//Load the script and execute it.  
String fileName = "TestStat2.py";  
String language = manager.getLangFromFilename(fileName);  
String script = IOUtils.getStringFromReader(new FileReader(fileName));  
manager.exec(language, fileName, 0, 0, script);
```

The manager's exec takes the same arguments as the eval method, the only difference being that the exec method doesn't return a value, i.e., it returns void. Notice that the scripting language can be decided based on the filename of the script by using the manager's getLangFromFilename method.

I have showed how to register a language, map application scriptable objects and execute a script. You can easily change the above to work with JavaScript, NetRexx and so on.

Parting Shots

The above example just scratches the surface of what BSF currently supports and what it will support in the future. Today IBM uses BSF with WebSphere to provide JSP development

Cimmetry Systems, Inc.

www.cimmetry.com

Developer- mentor

www.developermentor.com

that can be done in other languages like Python, TCL and JavaScript. In the near future it will be used with the Apache Tomcat servlet engine to support JSP for other languages, too. If you need to integrate a scripting language in your product, you should use BSF instead of picking a single scripting language.

I hope to see BSF used in a lot of development tools like IDE, editors and modeling tools so that the developers can easily customize their environment.

Java has wonderful features that make creating scripting languages easy. The class reflection and bean introspection APIs are a great basis for integrating scripting languages. Once the scripting language has metadata about a Java's class properties, events and methods, it can use this data to change properties, handle events and invoke methods. BSF complements this feature by providing you with a common way to map Java objects to scripting languages and a common interface for integrating your application with a multitude of scripting languages.

Components (JavaBeans) and distributed components (e.g., CORBA, EJB and RMI) have a symbiotic relationship with high-level languages. For example, Visual Basic did well because of VBX, OCX and ActiveX components and COM/ActiveX/DCOM did well because of tools like Visual Basic, PowerBuilder and Delphi. On the Java platform we have the component models, but we need the glue, i.e., tools for the high-level languages – debuggers, IDEs and so on. BSF provides a common way to “glue” components into applications using a multitude of scripting languages. BSF does for JavaBeans what MASF does for ActiveX controls. 🍌

AUTHOR BIO

Rick Hightower currently works at Buzzeo Corporation (www.buzzeo.com), the maker of ZEOLogix, an EJB application server, rules engine and workflow. In addition to being a principal software engineer working on an EJB container implementation and distributed event management, he is author of a book, Programming the Java APIs with JPython, to be published by Addison Wesley.

rick_m_hightower@hotmail.com

Listing 1

```
package stat;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.Collections;
import java.util.HashMap;

public class Stats {

    public ArrayList nums;

    public Stats(ArrayList someNums){
        nums = new ArrayList(someNums);
    }

    public Stats(double[] someNums){
        nums = new ArrayList(someNums.length);

        for (int index=0; index < someNums.length; index++){
            nums.add(new Double(someNums[index]));
        }
    }

    public double getMean (){
        return this.getMean(true);
    }

    public double getMean (boolean sample){
        // Define mean that finds two types of mean, namely:
        // population mean and sample mean
        double sum=0.0;
        double average=0.0;
        Iterator iterator = nums.iterator();

        while(iterator.hasNext())
            sum = sum + ((Double)iterator.next()).doubleValue();

        // Check to see if this is a sample mean
        if(sample)
            average = sum / nums.size()-1;
        else
            average = sum / nums.size();

        return average;
    }
}
```

Embar
cadero
p/u
www.
embarcadero.
com

```

public ArrayList getRange (){
    // Find the range. Returns a tuple with the minimum,
    // maximum, and range value
    double min, max;
    ArrayList ranges;

    min = ((Double)Collections.min(nums)).doubleValue();
    max = ((Double)Collections.max(nums)).doubleValue();

    ranges = new ArrayList();
    ranges.add(new Double (min));
    ranges.add(new Double (max));
    ranges.add(new Double (max-min));

    return ranges;
}

public double getMedian (){
    // Find the Median number

    // create a duplicate since we are going to modify the
    // sequence
    ArrayList seq = new ArrayList(nums);

    // sort the list of numbers
    Collections.sort(seq);

    double median = 0.0; // to hold the median value

    int length = seq.size(); // to hold the length of the
    // sequence

    int index=0;

    // Check to see if the length is an even number
    if ( ( length % 2) == 0){
        // since it is an even number

```

```

        // add the two middle number together
        index = length / 2;
        double m1 = ((Double)seq.get(index-1)).doubleValue();
        double m2 = ((Double)seq.get(index)).doubleValue();
        median = (m1 + m2) /2.0;
    }
    else{
        // since it is an odd number
        // just grab the middle number
        index = (length / 2);
        median = ((Double)seq.get(index)).doubleValue();
    }
    return median;
}

private int countMode(Object object, ArrayList list){
    int index = 0;
    int count = 0;
    do {
        index = Collections.binarySearch(list, object);
        if(index >=0)list.remove(index);
        count++;
    }
    while (index >=0);
    return count;
}

public double getMode (){
    // Find the number that repeats the most.

    // make a duplicate copy of the nums argument
    ArrayList duplicate = new ArrayList(nums);

    Collections.sort(duplicate);
    double highest_count = -100;
    double mode = -100;

```

QuickStream Software

www.quickstream.com

JD Store.com

www.jdjstore.com

JD Store.com

www.jdjstore.com

Pervasive, IBM and Caldera Systems Deliver Linux-Based E-Business Bundle

(Hong Kong) – Pervasive Software, Inc., is teaming up with IBM and Caldera Systems to deliver a Linux-based e-business bundle for the Korean Web application developer market, combining Pervasive and Caldera software products on IBM servers.

Available in Korea through distribution from Daesang Information, the bundle is designed for Web application developers in the Korean market delivering Linux-based e-business solutions. The IBM server-based bundle combines the Web application-enabling features of Tango 2000 and Caldera's OpenLinux 2.3 into a complete hardware/software Web application development platform. ☺

www.pervasive.com

Introducing FioranoMQ

(Los Gatos, CA) – Fiorano, Inc., a leading provider of standards-based, Java technology messaging solutions, has released the FioranoMQ Bridge for MSMQ and IBM MQSeries, enabling Java applications to leverage an existing enterprise messaging infrastructure to send and receive messages using the JMS API. A free development license for FioranoMQ including the bridging technology is available for download from the Fiorano Web site. ☺

www.fiorano.com

Intercontinental Exchange Selects Gemstone's E-Business Software

(Beaverton, OR) – GemStone Systems, Inc., has announced that its e-business application server software was chosen as the foundation for IntercontinentalExchange's B2B Internet trading platform for globally traded over-the-counter commodities.



GemStone's e-business solution – with Extreme Clustering technology – will enable IntercontinentalExchange to provide its customers with a real-time trading exchange for increased market transparency, liquidity and secure transactions. Multiple traders will more efficiently access, share and retrieve huge amounts of data and information simultaneously. ☺

www.gemstone.com

CodeMarket Stocking Its Shelves

(New City, NY) – CodeMarket is accepting submissions from individual developers for their online Java object marketplace. With the aim of becoming the premier destination to buy and sell unit-tested Java objects, CodeMarket uses obfuscation technology to offer objects and components on



a free trial basis to buyers. Sellers receive free listings and testing and get to work with CodeMarket technicians to improve their code documentation, pricing and licensing. For details of a current promotion including 70% commissions to sellers and bonuses for referrals, go to www.codemarket.net/specialoffer. ☺

Metrowerks Releases CodeWarrior with PersonalJava Support

(Austin, TX) – Metrowerks has announced the availability of CodeWarrior, PersonalJava Platform Edition, Version 1.0, which allows developers to create applications and content for high-end networked consumer devices such as mobile handheld computers, set-top boxes, Internet screen phones and Internet appliances that support the PersonalJava platform and API.



Using the award-winning CodeWarrior IDE, developers will be able to add valuable content to any networked consumer device that executes PersonalJava virtual machine and class libraries from a LAN or wireless network. ☺ www.metrowerks.com

Buzzeo Appoints JDJ Writer to Manager of ZeoLogix

(Phoenix, AZ) – Buzzeo recently announced the appointment of Rick Hightower as manager of ZeoLogix, Buzzeo's rules-driven, CORBA-based business information tool.

A self-described C++ bigot (“Programming Languages for the JVM,” *Java Developer's Journal*, Vol. 5, issue 4), Hightower learned, albeit reluctantly at first, that Java improved his development



SYS-CON Media Expands

(Montvale, NJ) – SYS-CON Media, Inc., publisher of *Java Developer's Journal* and other E-Business/Internet and Web-related magazines such as *XML-Journal*, has relocated its worldwide corporate headquarters from Pearl River, New York to nearby Montvale, New Jersey.

SYS-CON now occupies space in the 66,000 square-foot building pictured at left, situated in prestigious northern Bergen County, New York City is just a short drive away via the George Washington or Tappan Zee bridges, and the new location is convenient to Newark International Airport via a local shuttle service.

Prominent corporate neighbors within a half-mile distance include Sony, Mercedes and Volvo (U.S. headquarters), A&P and Grand Union (world headquarters), and other national and international corporations. ☺ www.sys-con.com

Wintertree

www.wintertree-

[software.com](http://www.wintertree-)

team's productivity. Since that epiphany, Hightower shifted his focus to Java and prior to joining Buzzeo was a senior software engineer specializing in Java architecture for Intel. ☞

www.buzzeo.com

SilverStream Ships jBroker 3.0

(Burlington, MA) – SilverStream Software, Inc. has unveiled SilverStream jBroker version 3.0, a Java-based Object Request Broker (ORB) and services specifically designed for building high-performance, e-business applications using the latest CORBA and J2EE technologies.

jBroker 3.0 is fully CORBA 2.3 compliant, with important features such as Java Portable Object Adapter (POA) mapping, Remote Method Invocation (RMI) over Internet InterOrb Protocol (IIOP), IIOP over Secure Sockets Layer (IIOP/SSL), Interoperable Name Service, and Objects by Value. A free 45-day trial version of the jBro-

ker Server can be downloaded from the SilverStream Web site. ☞

www.silverstream.com

eXcelon Corporation and Oracle Form Alliance

(Burlington, MA) – eXcelon Corporation and Oracle Corporation have announced an agreement through which the companies will co-market a set of products that will accelerate the use of XML in Java application development environments.

Under the terms of the agreement, eXcelon and Oracle will market eXcelon Stylus 2.0 in conjunction with Oracle's JDeveloper 3.1. When used together, the companies' products greatly enhance the Java development and deployment environment for creating enterprise-scale, high-performance e-business applications. ☞

www.exceloncorp.com

WebGain Acquires TOPLink from The Object People

(Cupertino, CA) – WebGain, Inc., the independent entity formed by

Warburg Ventures and BEA Systems, Inc., has announced it will acquire TOPLink, one of the industry's most advanced Java object-to-relational mapping products, from The Object People.

TOPLink is a key technology in WebGain's flagship product, WebGain Studio Professional, an end-to-end browser-to-database development solution comprised of leading technologies acquired and licensed from Symantec (VisualCafe), Macromedia

(Dreamweaver), Tendril Software (StructureBuilder) and Sun (Java).

TOPLink will continue to be sold both separately and as part of the WebGain Studio Professional product. ☞

www.webgain.com

Bluestone Software Charts Future of Total-e-Business

(Philadelphia, PA) – Bluestone Software, Inc., has announced its roadmap charting the future of Total-e-Business (TeB), the company's comprehensive, standards-

based e-business platform. It identifies five TeB Platform editions as well as the new Syndication Server, all designed to address specific e-business require-

ments by combining the required product functionality with a solutions-oriented understanding of customer needs. ☞

www.bluestone.com

Unify Acquires New Atlanta

(San Jose, CA) – Unify Corporation has acquired privately held New Atlanta Communications LLC, a premier provider of Java

Servlet and JSP technology, for an undisclosed amount of Unify stock. The acquisition allows Unify to leverage New Atlanta's leading server-side Java technology, extending Unify's momentum as a principal provider of open, component-based e-commerce solutions. ☞

www.newatlanta.com

www.eWaveCommerce.com

North-woods

www.nwoods.com

Visualize

www.visualizeinc.com



Interview...

with **ALAN ARMSTRONG**

JPROBE PRODUCT MANAGER KL GROUP INC.



JDJ: Tell us what KL Group has been up to since we last spoke with you at JavaOne in 1999.

Armstrong: Lots. We've been very busy over the past year. Now, with the release of version 2.8, JProbe is reinforcing its leadership in the advanced-development tools market by building on the server-side advantage to include speed and ease of use. In September we announced the JProbe ServerSide Suite, which gives developers powerful performance profiling, memory debugging, code coverage and thread analysis capabilities in one integrated suite. Designed for server-side Java development, JProbe ServerSide has been a tremendous success and a leader in the field. It's the most powerful and accurate tool of its kind. Coming up this month we're excited to announce the release of JProbe 2.8, introducing new features such as the Garbage Monitor, Detail Meter and dramatic performance improvements, plus there's also a Quick Start Wizard to guide less experienced developers through the tuning process.

JDJ: How does that differ from JProbe for the normal client site?

Armstrong: Much of it is very similar in that you're still analyzing code, but we've actually introduced a server launch pad that allows you to select from a list of our supported application servers. Also, it's an extensible list that allows you to add your own application server or servers that may not be "in the box." It basically reconfigures JProbe so that it automatically starts up the application server and makes it a breeze to tune servlets, EJBs and other server-side applications.

JDJ: One of the interesting things you explained last time we talked was how Java applications tend to have memory leaks.

Armstrong: Yes. One of the misconceptions about Java is that there are no such things as memory leaks. We've actually done a lot of work in classifying the kinds of memory leaks that can occur in Java. What it amounts to basically is memory that's never released by your application, so there's no way for the garbage collector to know it's actually garbage and thus reclaim it as part of the heap. It has to do

Armstrong: That's a good question. Memory leaks are actually independent of the implementation of the virtual machine. There are some changes with the HotSpot technology and how that handles garbage collection, but the fact remains that if there's a loitering object, in Java terminology, it's going to be a loitering object no matter what garbage collection scheme you are using. It would be as if I said, "We just got a better office cleaner, but you've still got a messy desk there." If you've got a messy desk, it doesn't matter how good the office cleaner is, you've got a messy

important, and you consider features to be more important than performance or reliability, then maybe it would be okay to leave it to the end of the project. But to be honest with you, we have a lot of customers who leave it that long, and no matter how good JProbe is at showing them where the problems are, it can be very expensive to fix them after the fact. Our most successful customers are those who make this tool available to all of their developers and do so much earlier in the development process.

JDJ: So your advice is to be working from an evolution point of view with your project.

Armstrong: Yes.

"Our most successful customers are those who make this tool available to all of their developers and do so much earlier in the development process"

with what we call *loitering* objects – objects that are allocated and reference to them is held on to far longer than it should be, so that memory is never reclaimed.

Now the interesting thing about memory leaks in Java – and by the way, the JProbe memory debugger is an excellent way to find those loitering objects and those memory leaks – is that typically, while less common than they were in C or C++, when they do occur they can be much more severe. That's because one object can hold references to thousands of objects and essentially slow down an application tremendously or even cause it to crash.

JDJ: Other virtual engines are available besides the standard one from Sun. Is any one in particular best at handling, for example, the memory leaks?

desk. The same applies in Java applications: if you've got memory leaks, you've got memory leaks.

JDJ: What's the procedure for using JProbe? Should I wait until I finish my application to start using one of my analysis tools, or should it be something I evolve with you hand in hand?

Armstrong: It really boils down to your philosophy about development...to how important performance and reliability are to the particular project you're developing. If you consider them to be low risk, not

JDJ: Is this at the class level – once you've concluded a class and are fairly happy with its implementation, you then run JProbe with that class?

Armstrong: We recommend doing it on a regular basis with your application and there's a right time to start doing it. You know, I'm not going to say you need to analyze every class you develop right from the very start, but pick the time when you want to start testing a set of classes or a component of your application – that might be a good time to start. Then again, it depends on the scale of your application and the importance of performance and reliability. ☉

XML DevCon 2000

www.xmldevcon2000.com

XML DevCon 2000

www.xmldevcon2000.com

XML DevCon 2000

www.xmldevcon2000.com

XML DevCon 2000

www.xmldevcon2000.com

XML DevCon 2000

www.xmldevcon2000.com

XML DevCon 2000

www.xmldevcon2000.com

XML DevCon 2000

www.xmldevcon2000.com

XML DevCon 2000

www.xmldevcon2000.com

XML DevCon 2000

www.xmldevcon2000.com

XML DevCon 2000

www.xmldevcon2000.com

XML DevCon 2000

www.xmldevcon2000.com

XML DevCon 2000

www.xmldevcon2000.com

XML DevCon 2000

www.xmldevcon2000.com

XML DevCon 2000

www.xmldevcon2000.com

XML DevCon 2000

www.xmldevcon2000.com

Career Opportunities

Career Opportunities

Career Opportunities

Career Opportunities

Career Opportunities

Career Opportunities

Career Opportunities

Career Opportunities

Career Opportunities

Career Opportunities

Career Opportunities

Career Opportunities

Career Opportunities

Career Opportunities

Career Opportunities

Career Opportunities

Career Opportunities

Career Opportunities

Career Opportunities

Career Opportunities

Career Opportunities

Career Opportunities

Career Opportunities

Career Opportunities

Career Opportunities

Career Opportunities

Career Opportunities

Career Opportunities

Career Opportunities

Career Opportunities

Career Opportunities

Career Opportunities

Career Opportunities

Career Opportunities

When the Internet Gets Really Big...

Sooner or later *everything* will be connected

WRITTEN BY BRUCE SCOTT



The Internet is amazing. In just six short years it has spawned thousands of new businesses and generated billions of dollars of wealth. Dot-com fever has captured the hearts of America's technologists and entered the lives of many Americans. In the midst of all of this, it's useful to look at where we've come from, where we are now and where we're going.

Let's go back to the beginning of the PC era. In the early 1980s for the first time computers were small enough, cheap enough and simple enough to make their way to the desktops in homes and businesses. At home people used computers for personal finances, word processing and games. At work people used computers to access the business systems once directly accessible only to computer professionals. With these PCs and local area networks we saw the tremendous growth of the client/server industry and the growth of technology giants like Microsoft, Oracle, Sun and Intel. We also saw the emergence of one dominant access platform, Wintel.

In 1994, with the invention of the Internet browser by Netscape, we saw the birth of the Internet as we know it today. The funny thing about the emergence of the Internet era is that fundamentally it was built on the same technology as the PC and client/server era. Yes, we had computers and networks that were bigger, faster and cheaper, but the basic technology was the same as we saw in the client/server era. The main difference is that the IT industry became enlightened. With the Internet we discovered that if we agreed on certain standard ways of doing things, we could provide greater access to computer systems than ever before and thus grow the industry to everyone's benefit and profit. No longer did we need to debate how to send a byte from one computer to another. The debate was settled and now we send bytes using TCP/IP. Thus we see the emergence of a small set of key Internet standards such as TCP/IP, HTTP, HTML, Java and SQL. More Internet standards will emerge, such as XML and CORBA, but they'll emerge based on general industry acceptance, not on the agenda of one or two major industry players.

Another thing the Internet has brought us is the beginning of the end of the Wintel dominance as the access platform. Because of the industry-wide agreement on open standards, we're discovering that Wintel isn't required to access the Internet. The best example of this is the iMac. Here's a technology that's been around for over 10 years and has seen resurgence just because Apple repackaged it as an Internet access platform.

So where are we going now? Most experts will agree, I think, that when history is finally written it will mark year 2000 as the beginning of the Post-PC Era. And what will this era bring us? Computers will penetrate every aspect of modern business and personal life. Post-PC Era computers will be everywhere. We'll come to depend on them as we depend on electricity. Corporations will be able to extend their reach to customers and employees everywhere and at any time.

Primitive single-function devices such as cell phones will become sophisticated application platforms. Everything will be connected to the Internet at one time or another. Data and information will be available 24x7, anywhere, anytime. There will be many, many varieties of access platforms and the Wintel dominance will fade into history. Java will be the technology that will enable the deployment of a single application on the plethora of Post-PC Era platforms.

The mobile Internet creates some new challenges. Connect time is expensive and although it's getting cheaper, there will always be a cost, hidden or otherwise. Also, wireless connect time is the biggest drain of battery power. People using Post-PC Era computers will want 24x7 access to their critical applications and won't want to wait until cellular

connectivity just happens to be available, nor will they tolerate the interruption of critical activities as a result of unreliable connectivity. This drives the need for mobile wireless applications to function while not connected. Activities on mobile wireless devices will generate data that eventually must be synchronized with back-end systems. This drives the need for local storage of data on mobile wireless devices. At PointBase we are the providers of the

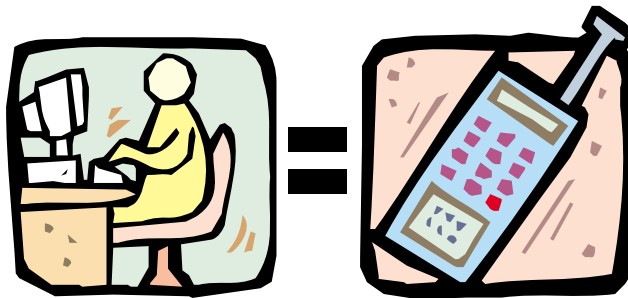
data management and data synchronization products that enable creation of mobile wireless applications that are available anytime, anywhere. PointBase is simple, small and 100% Pure Java.

The amazing growth of the Internet era has been fueled by a computer on a desk that is attached to the wall. The assumption has been that the center of productivity and creativity is around a stationary desk in your home or office. What happens if you can get the same level of productivity and creativity anywhere, anytime? This is when the Internet will be unleashed and become *really* big. If you think it's big now, just wait and watch as the Post-PC Era unfolds. ☪

AUTHOR BIO

Bruce Scott, president, CEO and founder of PointBase, is a leader in the area of enterprise and embedded database architecture and product development. A cofounder of Oracle in 1977, Bruce cofounded Gupta Technology in 1984, pioneering the notion of the small-footprint database server for Intel-based platforms.

bruce.scott@pointbase.com



SilverStream

www.silverstream.com

KL Group Inc

www.klgroup.com/collect