THE J2EE AND E-BUSINESS

JAVA™ DEVELOPER'S JOURNAL

The World's Leading Java Resource

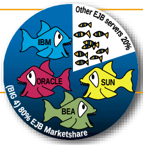
Volume: 4 Issue: 12, December 1999

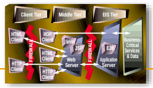
JAVADEVELOPERSJOURNAL.COM

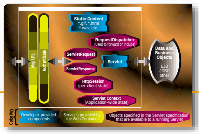
JDJ EXCLUSIVE! JAVA CAREER OPPORTUNITIES SECTION P.87 JOBS $

RETAILERS PLEASE DISPLAY
UNTIL FEBRUARY 29, 2000

SYS-CON PUBLICATIONS







JAVA DEVELOPERS JOURNAL

EDITORIAL ADVISORY BOARD
TED COOMBS, BILL DUNLAP, DAVID GEE, MICHEL GERIN,
ARTHUR VAN HOFF, JOHN OLSON, GEORGE PAOLINI,
KIM POLESE, SEAN RHODY, RICK ROSS,
AJIT SAGAR, RICHARD SOLEY, ALAN WILLIAMSON

EDITOR-IN-CHIEF:	SEAN RHODY
EXECUTIVE EDITOR:	M'LOU PINKHAM
ART DIRECTOR:	ALEX BOTERO
PRODUCTION EDITOR:	CHERYL VAN SISE
ASSOCIATE EDITOR:	NANCY VALENTINE
EDITORIAL CONSULTANT:	SCOTT DAVISON
TECHNICAL EDITOR:	BAHADIR KARUV
PRODUCT REVIEW EDITOR:	ED ZEBROWSKI
INDUSTRY NEWS EDITOR:	ALAN WILLIAMSON
E-COMMERCE EDITOR:	AJIT SAGAR

WRITERS IN THIS ISSUE
DEREK C. ASHMORE, SCOTT DANFORTH, ARNY EPSTEIN, GREG FLURRY,
HARRY FOXWELL, JIM MILBERY, SEAN RHODY, AJIT SAGAR, JON SIEGEL,
SAM WATTS, JASON WESTRA, ALAN WILLIAMSON, WILLIAM WRIGHT

SUBSCRIPTIONS
FOR SUBSCRIPTIONS AND REQUESTS FOR BULK ORDERS,
PLEASE SEND YOUR LETTERS TO SUBSCRIPTION DEPARTMENT
SUBSCRIPTION HOTLINE: 800 513-7111
COVER PRICE: $4.99/ISSUE
DOMESTIC: $49/YR. (12 ISSUES) CANADA/MEXICO: $69/YR.
OVERSEAS: BASIC SUBSCRIPTION PRICE PLUS AIRMAIL POSTAGE
(U.S. BANKS OR MONEY ORDERS). BACK ISSUES: $12 EACH

PUBLISHER, PRESIDENT AND CEO:	FUAT A. KIRCAALI
VICE PRESIDENT, PRODUCTION:	JIM MORGAN
VICE PRESIDENT, MARKETING:	CARMEN GONZALEZ
CHIEF FINANCIAL OFFICER:	IGNACIO ARELLANO
ACCOUNTING MANAGER:	ELI HOROWITZ
CIRCULATION MANAGER:	MARY ANN MCBRIDE
ADVERTISING ACCOUNT MANAGERS:	ROBYN FORMA MEGAN RING
JDJSTORE.COM:	JACLYN REDMOND
ADVERTISING ASSISTANT:	CHRISTINE RUSSELL
GRAPHIC DESIGN INTERN:	AARATHI VENKATARAMAN
WEBMASTER:	ROBERT DIAMOND
WEB EDITOR:	BARD DEMA
WEB SERVICES CONSULTANT:	BRUNO Y. DECAUDIN
WEB SERVICES INTERN:	DIGANT B. DAVE
CUSTOMER SERVICE:	SIAN O'GORMAN ANN MARIE MILILLO
ONLINE CUSTOMER SERVICE:	AMANDA MOSKOWITZ

EDITORIAL OFFICES
SYS-CON PUBLICATIONS, INC.
39 E. CENTRAL AVE., PEARL RIVER, NY 10965
TELEPHONE: 914 735-7300 FAX: 914 735-6547
SUBSCRIBE@SYS-CON.COM

JAVA DEVELOPER'S JOURNAL (ISSN#1087-6944)
is published monthly (12 times a year) for $49.00 by
SYS-CON Publications, Inc., 39 E. Central Ave., Pearl River, NY 10965-2306.
Periodicals Postage rates are paid at
Pearl River, NY 10965 and additional mailing offices.
POSTMASTER: Send address changes to:
JAVA DEVELOPER'S JOURNAL, SYS-CON Publications, Inc.,
39 E. Central Ave., Pearl River, NY 10965-2306.

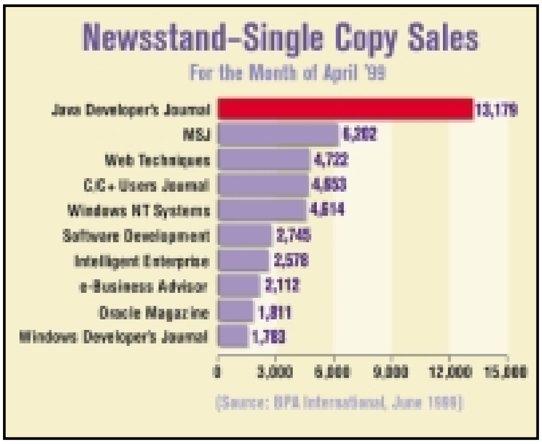
WORLDWIDE DISTRIBUTION BY
CURTIS CIRCULATION COMPANY
739 RIVER ROAD, NEW MILFORD NJ 07646-3048 PHONE: 201 634-7400

FUAT KIRCAALI, PUBLISHER, JAVA DEVELOPER'S JOURNAL

Java Developer's Journal Circulation
Reaches New Record High

We have exciting news for you. The qualified circulation of ***Java Developer's Journal*** print edition reached a record high of 70,017 for the June issue, according to BPA International's June 1999 initial audit report. For the June issue ***JDJ*** printed and distributed over 104,000 copies, including 13,000 bonus copies distributed at JavaOne in San Francisco. Show copies were not included in ***JDJ***'s qualified circulation figures. Based on the most current data available through BPA International (June statements), ***JDJ***'s qualfied circulation now is higher than *Java Report* and *Java Pro* combined.

Newsstand-Single Copy Sales
For the Month of April '99



(Source: BPA International, June 1999)

JDJ's single-copy sales also reached a record high in April among 10 leading consumer computer titles. According to BPA International, 13,179 copies were sold on the newsstands, an impressive net sales number – more than twice as many as *Microsoft Systems Journal*, *Web Techniques*, *C/C++ Users Journal*, *Windows NT Systems* and others.

I'd like to personally thank you, our readers, for making ***JDJ*** your favorite Java magazine since our premier issue more than four years ago.

Java Developer's Journal's target print circulation is 160,000 copies, and online circulation is 3 million quarterly page views. When the target numbers are achieved, ***JDJ*** will have the highest circulation of all consumer software titles audited by BPA International.

The last news I would like to share with you is that SYS-CON Publications, publisher of ***Java Developer's Journal*** was named as the fastest-growing, privately held publishing company in America. We ranked 194th in this year's *Inc. 500* list.

fuat@sys-con.com

SEAN RHODY, EDITOR-IN-CHIEF

The Past Through Tomorrow

It's not often you get to write an end-of-the-millennium column (once every thousand years, last time I checked). I thought that a little reminiscing about the past few years might be in order, followed by a brief look in the crystal ball to see what we have in store for you in the next century.

The first time I saw Java, it was 1996. Back then, few could imagine the impact this small, object-oriented language would have on the world. I was working in several 4GLs at the time and didn't immediately see the value of write once, run anywhere. As you can see, I've come around.

Java grew, slowly at a rapid pace. Sounds goofy, but when you think about it, it's taken years to get to where we are today, even if it seems that a new API or specification is released each week. We're in our third release of Java, counting 1.0, 1.1 and 2.0. We've overcome an inefficient event model and an ineffective GUI toolkit. We can now write good-looking applications that will run anywhere.

We've also come to the realization that Java is a good language for writing distributed applications, particularly the server side of things. Java 2 Enterprise Edition, with EJB, JSP and HTML, have made it possible to build fairly complex apps that require no client resident code.

Some things haven't worked out, such as the JavaOS and some of the thin-client Java hardware. But even that has had productive results, pushing vendors toward a Web-based model while contributing to the market competition that eventually drove down the price of PCs to a level that the average household can afford.

Java Developer's Journal has also grown through the years. Last year we presented our first Editor's Choice Awards to the vendors we thought created the finest products for Java, whether development environments, application servers or end products. This year we expanded that with our Readers' Choice Awards, in which you, our readers, were able to select your favorite products. We even spawned a new magazine, ***XML-Journal***. And we launched the **JDJ Store**, a place where Java developers can go to get the lowest prices on Java products (come visit us!).

And now it's time to look into the future. Java continues to improve and expand. I expect Jini to languish for the next year, then suddenly pick up steam as vendors begin to release products that support it. Ideally, Sun will make some arrangements with strategic hardware vendors to include this API as part of their support list.

I also expect to see Java make inroads onto portable devices. Some version of Java will have to appear on the PalmPilot. Phones and other tools will converge, and Java will be an important part of providing key services on these devices.

The rise of the Java Application Server will occur in the next year. Already we're seeing vertical products such as trading engines built on the Java 2 Enterprise Edition specification. This will continue, as vendors and ISPs have seen that the EJB approach is easier and more powerful than CORBA or DCOM.

Expect XML to play an increasing role in your Java future. This metalanguage is growing by leaps and bounds. Expect to see more parsers, new editors and perhaps even alternative serialization of classes via XML.

JDJ will also be busy in the future. Plans are underway to create a ***JDJ*** Laboratory to test various products and provide head-to-head comparisons. We expect to have the lab operational early in 2000, and are planning to showcase its power with an application server showdown featuring some of the best app server products on the market. We'll also be doing other types of product reviews in a comparative manner.

I expect that the Y2K lockdown that's been in place for the past few months will be removed after we go through January. Most companies are prepared; few will have disruptions. And all of that energy that's currently on hold will be spent addressing Web-based development. It will be a great time to be in the industry.

So as the year, the decade, the century and the millennium all draw to a close, we here at ***Java Developer's Journal*** wish you and your loved ones a happy, healthy and productive New Year. As always, thank you for making ***JDJ*** the number one Java magazine.

sean@sys-con.com

AUTHOR BIO

Sean Rhody is the editor-in-chief of Java Developer's Journal*. He is also a principal consultant with Computer Sciences Corporation, where he specializes in application architecture – particularly distributed systems.*

JDJ FEATURE

JavaBeans

Get the most horsepower out

WRITTEN BY WILLIAM WRIGHT

One of the great things about the JavaBeans specification is the flexibility it affords component developers in how they package their beans. As a bean developer, all you need is a class with a no-argument constructor that supports serialization and it's a bean. If you follow some simple naming conventions, most integrated development environments (IDEs) can tell enough about your bean to be able to use it for visual application development. While this is sufficient for some simple beans, the beans specification also provides ways for bean developers to explicitly give information that the IDE can use to assist the application developer in using the component. This article explores some of the ways that bean developers can make their components more usable by application developers through the use of custom property editors and customizers.

JavaBeans can be used by software development tools to allow visual programming using the beans as reusable components. This can reduce the amount of code that application developers need to write to make an application out of beans components. Beans are characterized by their properties and the events they generate and receive. Most beans have some properties that the application developer can configure during application construction to set the initial state of the bean. The Swing components are examples of beans with many properties that can be configured during application construction. They have properties like "backgroundColor" and "font" that control how the component looks and behaves. The bean developer needs only to follow some simple naming conventions when naming accessor methods to take advantage of this flexibility. If the bean has a property named "backgroundColor", a development tool will look for a method called "getBackgroundColor" that can be used to read the state of the property from the bean. If the bean has a method called "setBackgroundColor", it will be used to change the state of the property. Each of these methods is optional. That is, a bean can have a read-only property or a write-only property. Most beans are visual user interface components (like the Swing components), but beans can also be classes that are not visible on the user interface.

Development environments use Java introspection to interrogate a bean for its properties by looking for methods called get<something> and set<something> unless the property is a boolean; in that case the read accessor is called is<something>. With this information the IDE can build a graphical user interface called a customizer that the application developer can use to manipulate the bean. Often this is good enough. Simple beans may have properties that are of simple types and whose names are self-explanatory. Most IDEs include GUI components for Java primitive types and common classes like java.awt.Color and java.awt.Font, but if your bean has a property of a type that you defined, or isn't included in the IDE's set of property editors, it will just omit the property from the bean's automatically generated customizer. Also, more complicated beans may have dozens of properties – most of which are not important to the bean user – or have complicated dependencies that aren't visible through introspection. In these cases it would make the bean more useful if you could provide a simpler customizer that guided the user in how to configure the bean and helped prevent errors.

While the JavaBeans specification allows for IDEs to create customizers for beans using introspection, it also allows for bean developers to provide customizers to be used in place of, or in addition to, the default customizer. Bean developers have the flexibility to simply provide an editor for a single property to be used within the default customizer, or to provide a complete GUI customizer that can replace the default customizer altogether.

Customized of your JavaBeans components

Property Editors

There are several ways to provide a custom property editor to be used within the default customizer that the IDE generates by introspection. I'll explore the simplest here with an example and then move on to a full-blown customizer.

Let's make a simple bean that's a digital clock. Its only important property controls whether it displays the time as 12-hour time with an AM/PM designation or as 24-hour time. I'll call this boolean property "twentyFourHourFormat" and when the property is true, the clock will use 24-hour time; when it's false, it will use 12-hour AM/PM time. The code for this bean is in Listing 1.

Beans-aware IDEs have a property editor for the boolean primitive type; Figure 1 shows the customizer and bean display that the Sun BeanBox provides for the Clock bean.

The BeanBox is Sun's reference implementation of a bean container. It's a part of the Beans Development Kit (BDK) and is available at http://java.sun.com/beans. The BeanBox is not a full-blown IDE, but it's a good tool for testing beans, property editors and customizers. In this case the BeanBox used introspection to generate the customizer panel that includes not only the twentyFourHourFormat property but also all of the properties from the bean's base classes. Notice that the property editor for the twentyFourHourFormat property is a combo box of the values true and false. The Introspector also found a property called "running" because the bean has methods called isRunning and setRunning. The "running" property should be hidden from the application developer so that the clock runs continuously. I'll show how to hide this property from the application developer later in this article.

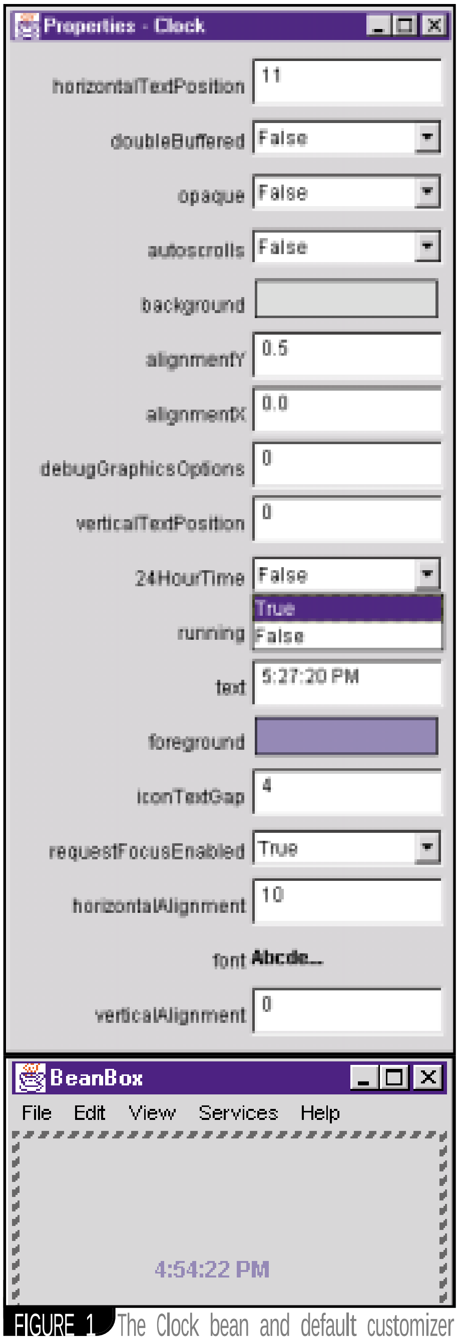
The automatically generated customizer isn't bad, but we can make it a little more user-friendly. Let's define a custom property editor that gives the user more information about what the property does.

All property editors must implement the java.beans.PropertyEditor interface. The easiest way to create a simple PropertyEditor is to extend the java.beans.PropertyEditorSupport class that implements java.beans.PropertyEditor and defines all of the interface's methods to reasonable defaults. Then I'll just need to override the methods that are necessary for the features I want to provide.

Instead of "true" and "false," it would be nice if the application developer could choose between "12 hour" and "24 hour" as the time format. I can do that with a custom property editor. If I override the getAsText() and setAsText() methods from PropertyEditorSupport, I can tell the IDE that the property can be set and read as a text string. If I also override the getTags() method, I can tell the IDE what the valid string values are for this property. The code for the property editor is shown in Listing 2.

Now the IDE can build a customizer that uses the strings that are in the property editor rather than true and false. But how does the IDE associate the property editor with the property? I'll need a BeanInfo class to make that association. A BeanInfo class provides information about a bean that isn't available through introspection. It can also be used to override the results of introspection. BeanInfo classes must implement the java.beans.BeanInfo interface. As in the case of the PropertyEditor interface, there is a class called java.beans.SimpleBeanInfo that implements all of the BeanInfo methods with reasonable defaults. I'll extend SimpleBeanInfo to define the ClockBeanInfo.

When examining a bean, an IDE looks for the BeanInfo class by appending "BeanInfo" to the name of the bean class and looking for a class by that name. If my bean is named mybeans.Clock, an IDE will look for the class mybeans.ClockBeanInfo that implements the BeanInfo interface. Ordinarily, a BeanInfo class goes in the same package with the bean it describes, but it can go in another package. IDEs will also look for BeanInfo classes in the packages returned by the static method in java.beans.Introspector called getBeanInfoSearchPath(). You can add your BeanInfo package to the search path with setBeanInfoSearchPath().



Properties - Clock

horizontalTextPosition	11
doubleBuffered	False
opaque	False
autoscrolls	False
background	
alignmentY	0.5
alignmentX	0.0
debugGraphicsOptions	0
verticalTextPosition	0
24HourTime	False True False
running	
text	5:27:20 PM
foreground	
iconTextGap	4
requestFocusEnabled	True
horizontalAlignment	10
font	Abcde...
verticalAlignment	0

BeanBox

4:54:22 PM

FIGURE 1 The Clock bean and default customizer

Thus, if your BeanInfo classes were all in the package mybeans.beaninfos, you could call this to tell an IDE about it:

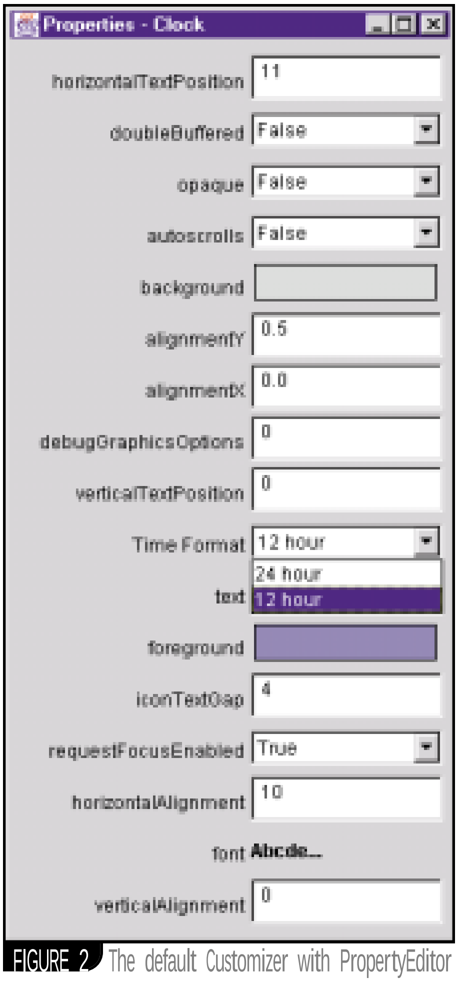
```
String [] path = {“mybeans.beaninfos”};
Introspector.setBeanInfoSearchPath(path);
```

If the IDE finds a BeanInfo for a bean, either in the same package as the bean or in the BeanInfo search path, it will ask the BeanInfo for information about the bean before using introspection.

An IDE uses the getPropertyDescriptors() method to get information about the bean’s properties from the BeanInfo class. getPropertyDescriptors() returns an array of java.beans.PropertyDescriptor objects. PropertyDescriptors define how the property should be displayed and edited. PropertyDescriptor and its superclass FeatureDescriptor also follow the beans naming convention for get and set accessors to properties. Table 1 summarizes the properties of a PropertyDescriptor.

The BeanInfo getAdditionalBeanInfo() method is used by the IDE to get properties from the ancestor classes of the bean. It relieves the BeanInfo class of the responsibility for creating PropertyDescriptors for all of the properties of all of the superclasses of the bean. I’ll include getAdditionalBeanInfo() here so the inherited Swing properties are also displayed in the automatically generated customizer. The code for the ClockBeanInfo class is in Listing 3. The BeanInfo class can also be used to associate a set of icons with the bean. IDEs can use the icons in their palette as a graphical representation of the bean. The getIcon() BeanInfo method takes a request from the IDE for a type (color or monochrome) and size (16 or 32 pixels square) of icon and returns a java.awt.Image object if an icon of the requested type is available. Constants representing the different icon types are defined in the BeanInfo interface. As we’ll see, different IDEs use different icon types so it’s a good idea to include a variety if possible. The getIcon() method is also listed in Listing 3.

Once the BeanInfo and PropertyEditor classes are written, I can load them into the BeanBox and



Properties - Clock

horizontalTextPosition	11
doubleBuffered	False
opaque	False
autoscrolls	False
background	
alignmentY	0.5
alignmentX	0.0
debugGraphicsOptions	0
verticalTextPosition	0
Time Format	12 hour 24 hour 12 hour
text	
foreground	
iconTextGap	4
requestFocusEnabled	True
horizontalAlignment	10
font	Abcde...
verticalAlignment	0

FIGURE 2 The default Customizer with PropertyEditor

see the results. Notice how the property editor for the twentyFourHourFormat property now uses the information from the PropertyDescriptor in the BeanInfo to make the editor user interface. The name of the property is now listed as “Time Format” rather than “twentyFourHourTime” and the combo box contains “12 Hour” and “24 Hour” rather than true and false. Because the “running” property descriptor hidden property was set to true in ClockBeanInfo.getPropertyDescriptors(), the property doesn’t appear in the customizer that the BeanBox generated (see Figure 2)

PropertyDescriptor Properties	Meaning	Example
name	The name of the property	“twentyFourHourFormat”
displayName	The name to be displayed to the application developer	“Time Format”
readMethod	The method to read the property	beanClass.getMethod(“getTwentyFourHourFormat”, boolean)
writeMethod	The method to write the property	beanClass.getMethod(“isTwentyFourHourFormat”, boolean)
shortDescription	Text that describes the purpose of the property	“Sets the clock to display either 12-hour time or 24-hour time”
propertyEditorClass	The class that is to be used to edit the property	mybeans.ClockPropertyEditor.class
bound	True if the class fires PropertyChangeEvents when this property changes	
constrained	True if the class fires VetoableChangeEvents when this property changes	
expert	True if this property should be hidden from novice application developers	
hidden	True if this property should be hidden from all application developers	
preferred	True if this property should be prominently displayed to application developers	

TABLE 1 PropertyDescriptor properties

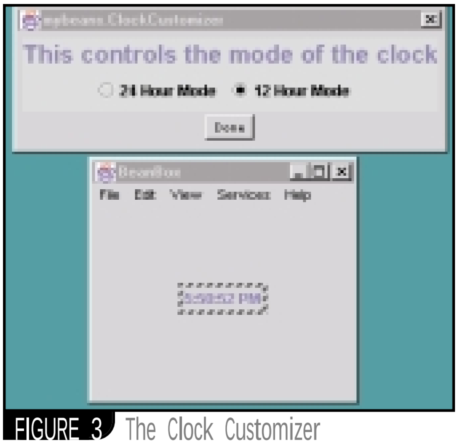


FIGURE 3 The Clock Customizer

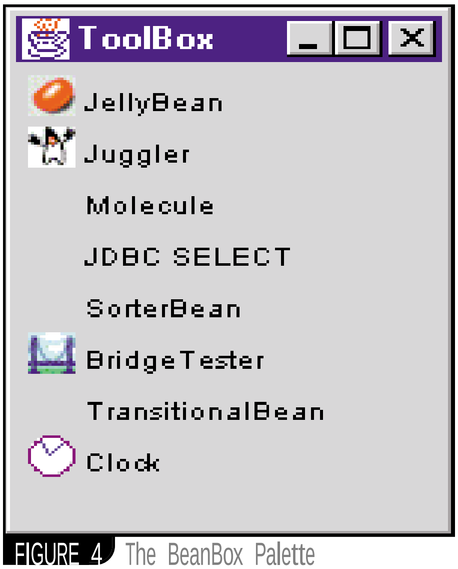


FIGURE 4 The BeanBox Palette

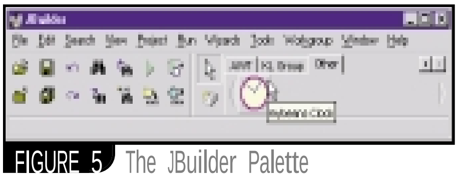


FIGURE 5 The JBuilder Palette

Customizers

By adding the BeanInfo and PropertyEditor, we've hopefully made the Clock bean a little easier to use in an application. I can take this one step further by asserting more control over the bean configuration process and define a customizer that can replace the default customizer generated by the IDE. By doing this I can show the user only the properties that are important and present the bean state in any way I want.

Like custom property editors, customizers are also associated with their beans through the BeanInfo class. The BeanInfo method getBeanDescriptor() returns a BeanDescriptor that contains the bean's class and the bean's customizer class if it exists. An IDE can use the bean descriptor to find out whether the bean's author has provided a customizer that the IDE can use instead of or in addition to the customizer it generates using BeanInfo and introspection. SimpleBeanInfo.getBeanDescriptor() asserts that there is no customizer, so to assert that there is one, I'll override getBeanDescriptor() in ClockBeanInfo in Listing 3.

Now I'll need to write the customizer. All beans customizers must implement the java.beans.Customizer interface and extend java.awt.Panel. The code for the simple Clock customizer is in Listing 4.

The IDE gives the customizer a reference to the to-be-configured object by calling the setObject method of the Customizer interface. The customizer can then synchronize its state with the object being customized, register as an event listener, etc. Then, as the application developer manipulates the customizer GUI, the bean can be updated immediately. This gives the application developer immediate feedback on what the effects are of changing a property.

To view the customizer in the BeanBox, first select the bean to be customized, then select View —> Customizer from the menu bar. Figure 3 shows what the customizer looks like in the BeanBox.

Packaging the Bean

Once the bean, BeanInfo, PropertyEditors and customizer have been written, they need to be packaged for loading into an IDE. The easiest way to load the bean and its associated classes and images into an IDE is to put them into a JAR file using a command like this.

```
jar cvmf manifest.mf
mybeans.jar mybeans\*.class mybeans\*.gif
```

The manifest file is an annotated list of the files that go into the JAR file. A special tag in the manifest file called "Is JavaBean" identifies the classes that are JavaBeans. The manifest file for this example is:

```
Manifest-Version: 1.0
Name: mybeans/Clock.class
Java-Bean: True
```

Every IDE loads beans a little differently, but I can go through a couple of examples here. All of the examples so far have used the BeanBox that comes with the Sun Beans Development Kit. Here's one way to load the JAR file into the BeanBox:

1. Under the File menu, select LoadJar.
2. Use the file dialog to select the JAR file that contains the beans. The BeanBox loads all of the beans in the JAR file into the BeanBox palette.

Figure 4 shows what the Clock bean looks like in the BeanBox palette. Note that it found the 16x16 color icon and put it in the palette with the bean class name.

A Real IDE

Borland's JBuilder uses a little different method but can load the same JAR file. These are the steps to load the Clock bean into the JBuilder 3 palette:

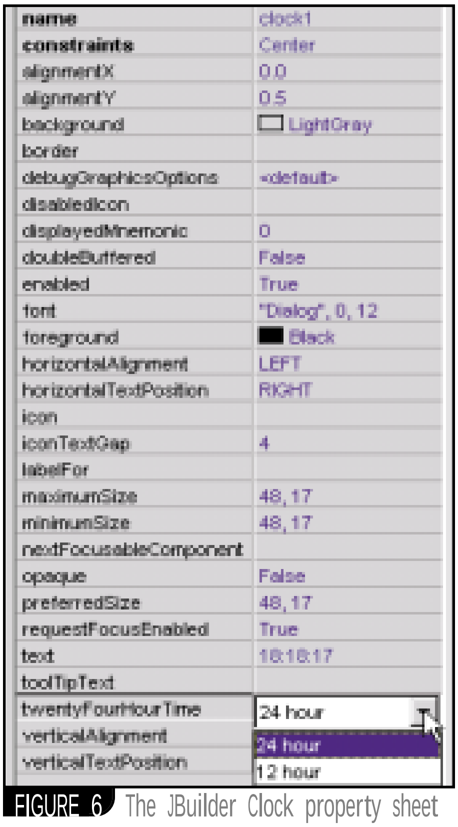
1. Select Tools —> Configure Palette.
2. Under the "Pages" tab, select the palette page to hold the component. A blank page called "Other" included in the default palette is a good place for new beans.
3. Select the "Add From Archive" tab and select the JAR file containing the bean and related classes. JBuilder opens the JAR file, reads the manifest and displays the beans that it found.
4. Select the bean class or classes to be installed.
5. Select "Install" to load the component.

Figure 5 shows what the component looks like in JBuilder. It looks like JBuilder preferred the 32x32 color icon to represent the clock bean in its palette.

Let's take a closer look at how JBuilder uses the Clock bean and its associated classes. If I drop the Clock from the palette onto a panel, JBuilder shows the clock in the panel and the Clock is running. JBuilder also creates a property sheet customizer containing the properties that I defined in the BeanInfo getPropertyDescriptors() method and the Swing properties that it found by using getAdditionalBeanInfo(). The Clock bean property sheet is shown in Figure 6.

IDEs have a lot of flexibility in how they use the BeanInfo information. Notice that JBuilder did use the property editor for the twentyFourHourTime property, but used the property name rather than the display name I set in the BeanInfo. It did pick up the short description text from the BeanInfo and used it as the ToolTip help text (see Figure 7). Cool. JBuilder also added two entries to the property sheet

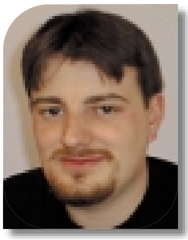
continued on page 80



name	clock1
constraints	Center
alignmentX	0.0
alignmentY	0.5
background	□ LightGray
border	
debugGraphicsOptions	<default>
disabledIcon	
displayedMnemonic	0
doubleBuffered	False
enabled	True
font	"Dialog", 0, 12
foreground	■ Black
horizontalAlignment	LEFT
horizontalTextPosition	RIGHT
icon	
iconTextGap	4
labelFor	
maximumSize	48, 17
minimumSize	48, 17
nextFocusableComponent	
opaque	False
preferredSize	48, 17
requestFocusEnabled	True
text	18:18:17
toolTipText	
twentyFourHourTime	24 hour
verticalAlignment	24 hour
verticalTextPosition	12 hour

FIGURE 6 The JBuilder Clock property sheet

Java Is for Life . . . Not Just for Christmas



WRITTEN BY
ALAN WILLIAMSON

Well, here we are again, decking the halls with boughs of holly, fa-la-la-la-la-la-la-la-la, and all that sort of nonsense.... The time of year when the opportunity to steal a kiss from the secretary isn't an actionable offense (assuming, of course, that you catch her – or him – under the mistletoe and not just as you're waiting for the printer to finish. So be careful).

It's been an exciting year, but I'm going to leave my reflection article to the next one, when it'll be the millennium and – with any luck – we should all be here in one piece. As we run up to the end of the century, I'm hearing reports that not all the predictions have come true. Our dear French prophet, Nostradamus, was way off base, unless world destruction has actually occurred and someone has neglected to inform us country boys. News does sometimes take a while to reach us, so who knows? I may be writing for a reader base that no longer exists.

I love Christmas. Such a wonderful time of year....A time when you make those agonizing decisions about who not to send a Christmas card to....About who has really annoyed you enough during the year to be taken off the list. Yes, Christmas...a most wonderfully cynical time of the year. And, in the words of a great English writer: "Bah, humbug."

A Sordid Tale

This month has seen us struggle with Oracle and Microsoft NT and the sordid tale will be recounted here in detail. One of our major clients has many an Oracle database running on a variety of Solaris platforms. We've written many solutions to interface and tie these databases together using Java Servlets. You all know the woes I've had in the past with JDBC-Oracle drivers and I can happily report that all our problems had gone away. The more astute of you will notice the word *had* in the previous sentence, which would suggest that the problem may have returned. Read on.

We had a spare NT server sitting around doing not much, and on a whim I had the bright idea of installing Oracle on it and using it as a development server, which would speed up our testing. So while I installed Oracle, I asked our client to do a complete export of the data, and the resulting 400MB file was transferred up to us. The version of Oracle the export had originated from was 8.0.3 (Solaris), and it was destined to end up in an 8.0.3 (NT) version. So no immediate problems were expected.

After a bit of buggering around with table permissions, the file imported with no errors reported. Excellent. Morale was high, and we proceeded to hook up our back-end server software and start some serious testing. Hah! If it were only that simple!

Our JDBC driver started throwing out exceptions immediately. "No more data to read from socket." What the hell sort of error is that? So we checked all our driver settings, making sure our database URL was correct and pointing to the right machine. No problems there. I had this real horrible sinking feeling that we were about to enter the world of JDBC-Oracle again. This world is a bit like the mythical Narnia: once you've entered through that wardrobe, it's very hard to get back out.

But it's a type 4 driver and runs on Solaris with no problems whatsoever. For those of you not too familiar with the types of JDBC drivers, a type 4 is a pure Java solution, which relies on no native code. Out of the four types, a type 4 is the most portable since it can run on any of the virtual machines. I was loath to even think of taking this route. We were running our software from a Linux configuration and even thought for a split second that this could be the cause of the error.

As a precaution, I downloaded and installed the latest JDBC driver. In a warped sort of way I was thankful it still didn't work. I didn't want to think it was the JDBC driver, for if it was I could easily be forgiven for thinking there were dark forces afoot that deemed I was never to have an easy time with Oracle and JDBC.

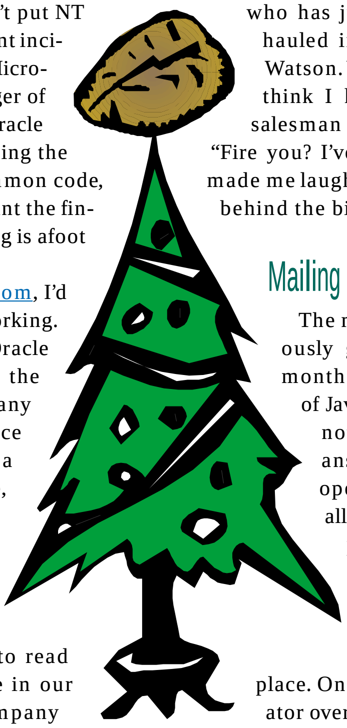
So what on earth could it be? Well, back to what is probably the best Web site in the world for more information. Deja.com is an archive of the daily postings to the newsgroups and can be quickly and easily searched using a diverse range of options. If you've never been there, go – it'll improve your problem-solving abilities tenfold. I dutifully put in the words "No more data to read from socket ORACLE," hoping to see at least one person who had met the same fate I had. Boy, was I surprised.

Hundreds of posts were made regarding this problem. Hot damn, now this is the true power of the Internet. I quickly located the fact that Oracle for NT has severe problems and I needed to upgrade to 8.0.5. Surely not. You mean Oracle is at fault at the core server? Now this surprised me. I had always been under the impression that Oracle was rock solid with version 8.0 and it was just their drivers that let them down. But after downloading the update, installing and converting over our database file, sure enough, our application burst into life. Fantastic.

I did some more digging and discovered that Oracle on NT has never been an exemplary piece of software. Many people, many problems. Now why is this? me wonders. I remember reading an American version of one of the computer magazines – I can't remember which one – but it had an article that talked up the use of NT in server environments. It explained why it was much better than a Linux alternative and really went to town to sell NT. It had to be written by a Microsoft employee. It completely contradicted every experience I've ever had with the two operating systems. But it put enough doubt in my head to go off and do a little reading.

After reading through many posts, I can't really say one way or another that any one operating system was getting more bugs than the other. It's very hard to tell. I can only comment from my own experience, which I have to say doesn't put NT in a very good light at all. For example, this recent incident with Oracle hasn't really strengthened Microsoft's case. As much as I'd love to point the finger of blame at Oracle, I can't…well, not fully. Their Oracle 8.0.3 is running perfectly on Solaris, so, assuming the NT version shares a significant amount of common code, the only difference has to be NT. I also can't point the finger of blame at Microsoft, but I think something is afoot that I'm not fully aware of.

Either way, without the assistance of Deja.com, I'd still have been struggling to get the bugger working. It has to be said that documentation on the Oracle Web site regarding this topic is very thin on the ground. For a company that's supporting many different versions, there should be a logical place where we can see all the problems. If such a page exists on the Oracle Web site, please, someone, e-mail me the URL. I thank you.



Book Review

I love reading books about the history of our great industry. It's a continual buzz to read about people who are still very much active in our world. I'm getting through many of these company biographies and one, which I haven't quite finished yet, is from Paul Carroll: *Big Blues: The Unmaking of IBM*. This is turning out to be a really fascinating read, discovering the ins and outs of one of the biggest companies in the world.

It contains a great story that Carroll retells. An executive who has just cost IBM $10 million in a failed deal is hauled into the office of then CEO/cofounder Tom Watson. Watson asks the sales executive, "Why do you think I have asked you here?" The rather worried salesman responds: "To fire me?" Watson then replies, "Fire you? I've just spent $10 million training you!" This made me laugh. But if you're ever interested in what goes on behind the big blue letters, this is a great insight!

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.

Mailing List

The mailing list is beginning to generate some seriously good threads of conversation. In the last month we've had a number of debates on the future of Java and whether it should be open sourced or not. One poster posed the question and we all answered. Surprisingly, not many supported the open sourcing of Java, which was good. Personally, I'd suspected a much greater swing of support, but it was good to hear everyone's structured answers about why it should, for the time being, be left to Sun to manage.

Discussions on what we'd like to see in Java during the next wave have also taken place. One topic that comes up time and again is operator overloading. I must say, I liked this facility in C++, and understand the apprehension of James Gosling not to include it in Java. But we hear that it's seriously being considered.

If you want to be part of the discussion, send an e-mail to listserv@listserv.n-ary.com with subscribe straight_talking-l in the body of the e-mail. From there you'll get instructions on how to participate on the list. Thank you all for your continued posts, and I have to say that I thoroughly enjoy the variety of topics discussed.

Salute of the Month

This month I'm going to honor not a person but a piece of equipment that for us here at n-ary has improved the quality of our lives tenfold. It's a device I'm sure has helped many others in their quest for cleanliness in the kitchen. The device I refer to, of course, is the humble dishwasher. The person who invented this beast should be knighted or declared a saint. We recently took delivery of one for our kitchen, and I tell you it's a nice feeling to go and make the coffee without having the sinking (no pun intended) feeling that you first have to wash a pile of dirty cups. So, dishwasher, we salute you.

Cool to Be Country?

Last month I told you about my foray into the world of Dolly Parton. Well, in keeping with the country theme, I was watching the Country Music Awards and was introduced to the Dixie Chicks. It was a wonderful tune and I instantly wanted more. So I went out and bought their latest CD and what a wonderful purchase that turned out to be. So I'm still in my country mode with no signs of letting up. If anyone has any advice on how to get out of this world, please let me know…sooner rather than later!

Until then, I bid you a happy holiday season and look forward to seeing you in the year 2000!

alan@sys-con.com

JDJ FEATURE

Strategies for Writing Java Stored Procedures for ORACLE Databases

WRITTEN BY Derek C. Ashmore

As of V8.i, Oracle developers can now write stored procedures, functions, packages and triggers in Java instead of PL/SQL (Oracle's proprietary procedural language), which provides some appealing options:

- We don't have to learn a proprietary (and thus limited-use) language to write stored objects for Oracle databases.
- We can get performance improvements over PL/SQL that make stored objects much more usable than they've been in the past.
- We can write code that's at least somewhat migratable to other database platforms should we wish to do so.

I'll provide guidelines and strategies for the effective use of Java within Oracle databases, and a brief overview of how to write stored procedures, functions, packages and triggers for Oracle Databases (V8.i) for readers who aren't familiar with these new features of Oracle. A basic knowledge of Java and Oracle database concepts (including SQL and the definitions of stored procedure, function, trigger and package) is assumed.

Java Stored Procedure Overview

REVIEW OF CAPABILITIES

Stored objects written in Java use JDBC to access Oracle databases the same way Java programs outside the database do. It's also common to embed SQLJ (a preprocessor that inserts generated JDBC code in your program) within Java stored procedures. The only coding difference with Java stored procedures is a change in how we initiate a database connection. There are also a couple of utilities that define Java programs to an Oracle database as well as changes to the CREATE PROCEDURE, FUNCTION, PACKAGE BODY and TRIGGER statements. If you're familiar with the capabilities of PL/SQL, stored objects in Java can do anything PL/SQL can do.

ORACLE JVM SPECIFICS

Because of Aurora's tight integration with the Oracle database kernel, it isn't pluggable. No upgrade to Java 1.2 was available at the time of this writing (though I'd expect one at some point).

LOADJAVA AND DROPJAVA UTILITIES

The two-step process to define Java stored objects is (1) load the Java class, and (2) expose its methods. Java classes are loaded into the database by either a CREATE JAVA statement or via the loadjava utility, which is typically executed from an operating system command prompt. In reality, the loadjava utility issues a CREATE JAVA statement behind the scene. I find the loadjava utility easier to use.

As an illustration, I've written a short class that'll determine a unique number for an identifier field of a table. Frequently, in Oracle-based applications, sequences that generate unique numbers are used to generate a unique number for use as a key field in a table. Unfortunately, the sequence that generates the number has no formal association with the field in the table using the number. This means programmers have to check for the possibility that the generated number isn't unique. My program centralizes this logic in one class so no one else has to code it (see Listing 1).

Java classes can be loaded as source, class or JAR files. Java source is compiled by the JVM in the Oracle database engine. The ability to load class and JAR files is nice because we can conceivably load purchased components (provided they aren't GUI components) into the database. An example of a loadjava statement follows:

```
loadjava -u derek/hello@venus:1521:ORA81a -thin -v -f -r -t
OracleProcs.java
```

The –v option produces detailed messages about the steps loadjava is going through to compile and load my Java class. The –f forces the loading of this Java class even though it's already present, which means I don't have to issue a dropjava command first. The –u option specifies the connection string in thin-driver format for the database in which this class is being loaded. The –r option designates that all external references are to be resolved at load-time instead of runtime. The –t option designates that the "thin" JDBC drivers are to be used for any database communication during the load process. The last argument specifies my Java source.

Similarly, I can remove my class with the dropjava utility. The command arguments are similar. The –v and –u options mean the same thing as with the loadjava utility. An example follows:

```
dropjava -u derek/hello@venus:1521:ORA81a -v -t OracleProcs
```

Once the classes have been loaded, we must expose individual methods with CREATE PROCEDURE, FUNCTION and PACKAGE BODY statements. It should be noted that the Java language libraries, JDBC libraries and ORB class libraries are already present in the database. No need to load them again.

CREATE PROCEDURE, FUNCTION, PACKAGE BODY AND TRIGGER STATEMENTS

While the loadjava utility will associate Java classes with the database, none of the methods associated with those classes are callable until you register them. Methods of Java classes are registered (or "wrapped") by issuing CREATE PROCEDURE, FUNCTION or PACKAGE BODY statements. After registration, they can be called in the same manner as PL/SQL procedures, functions and packages.

An example of a CREATE FUNCTION statement that registers a Java method is presented below:

```
create or replace function getID(
      TableName varchar2,
      ColumnName varchar2,
      SequenceName varchar2)
   return number
   as language java
name 'OracleProcs.getUniqueIdentifier(java.lang.String,
java.lang.String, java.lang.String) return double';
/
```

The "AS LANGUAGE JAVA" clause also works with CREATE PROCEDURE and CREATE PACKAGE BODY statements.

Note: You must fully qualify the argument passed if it isn't a native Java data type. As many of you know, strings aren't a native data type in the Java language. The definition of a string is obtained from the Java.lang import library. Hence, we must fully qualify the object type being passed.

Surprisingly, only a few alterations are needed to define a Java program as a stored object under V8.i. All Java stored objects use JDBC for database access.

JAVA REQUIREMENTS

All Java methods executed from a database connection must be declared as *static*. This makes sense as these methods are essentially being invoked from outside the JVM, and hence must be runnable (as method *main* always is). Once invoked, methods can instantiate nonstatic objects and use them, but these object allocations disappear after the method completes. If you wish to retain information within the Java class between method calls, you must store them in a static-defined variable.

Another major difference: within a Java stored object we initiate a JDBC connection differently. With stored procedures we'll typically use the connection created by the process that invoked the stored procedure as opposed to opening up a separate connection. An example of how to specify the default connection is:

```
Connection dbConnect = new OracleDriver().defaultConnection();
```

Issues to Be Considered When Introducing Java Stored Procedures

Consider the following issues before deploying Java stored procedures in your organization:

- *Platform compatibility requirements for your applications*
- *Developer training*
- *System management and source control*
- *Software licensing*
- *Application performance*

You need to identify the target Oracle database platform for your applications before migrating stored objects to Java. Java stored objects are new with V8.i; they're not supported in V8.0.5 and earlier. Applications that need to support earlier versions of Oracle software won't be able to migrate immediately.

As Java stored procedures use standard JDBC to issue SQL statements, Java is easier to migrate to other database platforms should that become necessary. I have a number of clients who would like to migrate an application from one database platform to another but can't because stored procedures are written in a proprietary, nonportable language. Stored procedures written in Java have a significant chance of being portable to a non-Oracle platform without a complete rewrite.

Developer training is usually a significant issue when adopting new technologies. For shops already developing in Java, introducing Java stored objects would be easy and inexpensive. No additional training would be necessary. For shops that don't use Java, training costs could be significant, but comparable to other languages. In addition, database administrators would also have to learn Java at a basic level in order to provide developer support.

Many Oracle environments use object ownership to distinguish between environments. It's common to define testing tables and indexes using one user ID and to create a development environment using another. For example, user DEV might own our development tables, indexes, and so on, while user TEST owns our testing environment in the same database. Oracle's JVM, like other JVMs, doesn't have a native ownership/object security model. Only one version of a class can be present in Oracle's JVM, so I can't have a development and testing version of the same class in a database. These environments must now be separated into different databases. Robust source control procedures are needed with Java stored procedures to avoid conflicts.

There's a software licensing issue with using Java stored procedures. At the time of this writing, Oracle's JVM was licensed separately and has its own cost component. From a strictly technical point of view, writing stored procedures in Java instead of PL/SQL has many advantages. However, the cost of Oracle's JVM may not be worth the benefits for some applications.

Performance Issues

Java stored procedures are much faster than PL/SQL. My tests indicated that Java stored procedures that do ordinary SQL statements, such as selects, updates, inserts and deletes, can be improved 20–40% if written in Java instead of PL/SQL. Additionally, procedures written in Java that don't issue SQL statements execute nine or 10 times faster than PL/SQL.

Java outperformed PL/SQL by 20–40% for SQL operations by allowing more flexible array processing and write batching. To get a simple SQL operation test, I wrote a Java and PL/SQL procedure to select and loop through all object names in the DBA_OBJECTS system view. For those that are interested, DBA_OBJECTS identified all objects existing in an Oracle database. At the time of my test there were 11,668 objects in my database.

Out of curiosity, I wrote the method to take the array size as an argument. Oracle allows array processing on select statements; array processing allows Oracle to retrieve rows in batches (e.g., 100 at a time) for efficiency. PL/SQL doesn't support array processing. Oracle's JDBC drivers set the array size to 10 by default. The Java source for this method can be found in Listing 2. The PL/SQL source for this method can be found in Listing 3. My results are in Listing 4.

The results show that by default (array size of 10) Java was about 18% faster than PL/SQL. However, if you employ array processing (which is easy to do with Java), you can get significant performance improvements for read operations.

As an aside, there are diminishing returns to increasing the array size – performance improves more if the array size is increased from 1 to 10 than from 100 to 200. To set the array size, use the setDefaultRowPrefetch method of the OracleConnection class. An example of how to do this is contained in Listing 1.

Strategies for Effectively Using Java Stored Procedures

Once you've decided to write stored procedures in Java instead of Oracle's native PL/SQL, you need to decide which Java classes should be deployed as stored procedures and which should be deployed normally (as part of an application, applet, CORBA service, etc.). The issues to think about when deciding whether to use Java stored procedures are:

- *JVM currency*
- *Application design*
- *Performance*

Oracle's JVM, Aurora, is currently in V1.1.6 and not pluggable. Java deployed as a stored procedure won't have access to features in later releases of the JVM (e.g., V1.2.x). While I'd expect Oracle to keep Aurora relatively current, it'll never be at the same level as the latest and greatest Java release.

From an application-design point of view, the fact that all methods called from a database connection need to be declared "static" tends to limit the role of Aurora to that of a "function loader." A class in this context is just an arbitrary collection of methods. While you can instantiate and use classes within a method call, any memory you allocate won't be available for future method calls. If you wish to retain information for future method calls, you must store this information in a statically defined and allocated variable. With this restriction it's hard to keep a purely object-oriented design for this section of the application.

Usage Guidelines

In my experience, Java code deployed as a normal application outside Aurora performs four to eight times faster than Java deployed as a stored procedure. Because of the various performance issues involved, I tend to deploy Java code outside the database as part of an application, applet, servlet, CORBA service, and so on. However, I do use Java stored procedures to implement the following items:

- *Database triggers*
- *Custom SQL column functions*

Database triggers execute code when INSERT, UPDATE or DELETE statements are issued. They're defined on a per-table basis. Triggers are used to enforce business rules that the database can't enforce via referential integrity constraints. For instance, I use triggers to execute the unique identifier generator that was reviewed in the first section of the article. An example of such a trigger definition follows:

```
create or replace trigger BEER_TR
   before insert on beer
   for each row
   when (new.beer_id is null)
   begin
    :new.beer_id :=
       getID('beer',
      'beer_id',
      'beer_seq');
   end;
/
```

Another place that Java stored procedures can be used effectively is in custom-column functions. Most developers are familiar with COUNT, SUM, AVERAGE and other native column functions that most databases provide. Using Oracle, it's possible to write custom column functions. I've used it in the past to format numbers (such as 999) into a currency format (such as $999.00).

Note: As Oracle Corporation is constantly tweaking its products, my usage guidelines for Java stored procedures may change for future versions of Oracle.

AUTHOR BIO

Derek Ashmore is the senior vice president of development for Delta Vortex Technologies, a Chicago-based consulting firm. He has designed, implemented and managed Oracle-based projects of many different types and sizes.

dashmore@dvt.com

Listing 1

```
//Title:       Utility Stored Procedures
//Version:
//Copyright:  Copyright (c) 1999
//Author:     Derek C. Ashmore
//Company:    Delta Vortex Technologies
//Description: Utilities which are callable
//             within the Oracle JVM

import java.lang.*;
import java.sql.*;
import oracle.sql.*;
import oracle.jdbc.driver.*;

public class OracleProcs {

  private static OracleConnection
  currentConnection = null;

  private static OracleConnection get-
  Connection(){
```

```
      return currentConnection;
  }

  private static void setConnection(
     OracleConnection oracleConnection)
            throws SQLException {
   currentConnection = oracleConnection;
   currentConnection.setDefaultRow-
   Prefetch(100);
   currentConnection.setDefaultExe-
    cuteBatch(20);
    currentConnection.setAutoCommit
    (false);
}

private static void setDefaultConnection()
           throws SQLException {
     setConnection((OracleConnection)
        new OracleDriver().defaultCon-
        nection());
}
```

```
public static double getUniqueIdenti-fier(
 String tableName,
 String columnName,
 String sequenceName)
  throws SQLException {
  double  uniqueID = 0;
  boolean idFound = false;
  OracleResultSet nextvalResults = null;
  OracleResultSet existResults = null;

  if (currentConnection == null)
      setDefaultConnection();

  // Select of sequence value
  String SQLStatement = "select " +
      sequenceName.toUpperCase() +
      ".nextval from dual";
  OraclePreparedStatement nextvalStmt =
     (OraclePreparedStatement)
    currentConnection.prepareState-
    ment(SQLStatement);
```

```
  // Verification of non-existence
  SQLStatement = "select count(*) from "
      + tableName.toUpperCase() +
      " where " + columnName.toUpper-
      Case() +
      " = ? and rownum <= 1";
      OraclePreparedStatement existStmt =
         (OraclePreparedStatement)
        currentConnection.prepareState-
        ment(SQLStatement);

  // Find next unused value
  while (!idFound)   {
     nextvalResults = (OracleResultSet)
       nextvalStmt.executeQuery();
     nextvalResults.next();
       uniqueID = nextvalResults.get
       Double(1);
          existStmt.setDouble(1,
          uniqueID);
          existResults = (OracleResult
          Set)
             existStmt.executeQuery();
     existResults.next();
       if (existResults.getInt(1) == 0)
           idFound = true;
     nextvalResults.close();
     existResults.close();
      }

      nextvalStmt.close();
      existStmt.close();

      return  uniqueID;
  }

}
```

Listing 2

```
import java.lang.*;
import java.sql.*;
import oracle.sql.*;
import oracle.jdbc.driver.*;

public class SqlTester {

 private static Connection nullConnect;

 private static Connection getConnection
     (int arraySize) {
    try {
    Connection conn = new
     OracleDriver().defaultConnection();
    ((OracleConnection)conn).setDefault-
    RowPrefetch
      (arraySize);
    return conn;
    }
    catch (SQLException sqlError) {
    System.out.println("This didn't
    work!");
    return nullConnect;
    }
 }
private static Connection getConnec-
tion() {
   return getConnection(1);
}

public static int performReadTest(int
arraySize) {
  String objectName;
  int objectCount = 0;

  try {
  Connection DBConnect =
  getConnection(arraySize);
  Statement sqlStmt = DBConnect.create-
  Statement();
  String sql = "select object_name from
  dba_objects";
  ResultSet dbResults = sqlStmt.execute-
  Query(sql);
  while (dbResults.next())  {
    objectName = dbResults.getString(1);
    objectCount = objectCount + 1;
  }
  dbResults.close();
  sqlStmt.close();
  }
  catch (SQLException sqlError) {
  System.out.println("We errored!");
  }
  return objectCount;
 }
 public static int performReadTest()
 {
  return performReadTest(1);
 }
}
```

Listing 3

```
create or replace function readtestplsql
return number is
 cursor OBJ_CSR is
   select object_name from dba_objects;
 H_OBJ_NAME  dba_objects.object_name%type;
 NBR_OBJECTS number := 0;
begin
 open OBJ_CSR;
 fetch OBJ_CSR into H_OBJ_NAME;
 while (OBJ_CSR%found) loop
    fetch OBJ_CSR into H_OBJ_NAME;
    NBR_OBJECTS := NBR_OBJECTS + 1;
 end loop;
 return NBR_OBJECTS;
end;
/
```

Listing 4

```
SQL> select readtestplsql from dual;

READTESTPLSQL
-------------
        11668

Elapsed: 00:00:03.40

SQL> select readtestjava(1) from dual;

READTESTJAVA(1)
---------------
          11668

Elapsed: 00:00:10.29

SQL> select readtestjava(10) from dual;

READTESTJAVA(10)
----------------
           11668

Elapsed: 00:00:02.78

SQL> select readtestjava(50) from dual;

READTESTJAVA(50)
----------------
           11668

Elapsed: 00:00:02.22

SQL> select readtestjava(100) from dual;

READTESTJAVA(100)
-----------------
            11668

Elapsed: 00:00:02.20

SQL> select readtestjava(200) from dual;

READTESTJAVA(200)
-----------------
            11668

Elapsed: 00:00:02.14

  public class Tester {

public static void counter(int inputVal-
ue) {
    int count = 0;
    while (count++ <= inputValue);
    }

public static int timeCounter(int input-
Value) {
   int timeElapsed;
   long timeMarking = System.current-
   TimeMillis();
   counter(inputValue);
   timeElapsed = (int) (System.current-
   TimeMillis()
      - timeMarking);
   return timeElapsed;
   }

public static void main (String argv[])
{
    Integer tempInteger = new
    Integer(argv[0]);
    int nbrIterations =
    tempInteger.intValue();
    long timeMarking = System.current-
    TimeMillis();

    counter(nbrIterations);
    System.out.println (System.current-
    TimeMillis()
       - timeMarking);
    }
}
  SQL> select timecounterjava(10000)
from dual;
TIMECOUNTERJAVA(10000)
----------------------
                     4
SQL> select timecounterjava(100000) from
dual;
TIMECOUNTERJAVA(100000)
-----------------------
                     42
SQL> select timecounterjava(1000000)
from dual;
TIMECOUNTERJAVA(1000000)
------------------------
                     420


  Java (Outside Oracle JVM) with JIT
enabled
->java derek 10000
1
->java derek 100000
6
->java derek 1000000
59
Java (Outside Oracle JVM) with JIT dis-
abled
->java derek 10000
2
->java derek 100000
27
->java derek 1000000
267
```

Santa Claus Is a CORBA Object

WRITTEN BY
JON SIEGEL

CORBA – the Common Object Request Broker Architecture – is an open, vendor-independent standard for software interoperability based on object technology. You've used CORBA even if you haven't heard of it by name:

- **It's the architecture of many of the best-scaling transaction processing systems.**
- **Virtually every application server on the market exposes CORBA interfaces.**
- **It's the basis of Enterprise JavaBeans interoperability.**
- **It's used to program many of the largest consumer e-commerce sites on the Web.**

CORBA from Both Sides: Client and Server

Applications' use of CORBA objects follows a number of different patterns, depending on what their objects represent or what they do. In this article we'll concentrate on persistent objects, objects that represent something long-lived in the real world – an account at a bank or brokerage house, or a tank in a military simulation, or the shopping carts in your e-commerce Web site. Once it's created an object instance and holds its object reference (a token that represents the object's address), a CORBA client can assume that the particular object instance continues to exist, maintaining its state all the while, and the reference remains valid until the client (or some authorized agent) explicitly destroys the object. Whenever it wants, the client can make an invocation using the reference and the answer will come winging its way back across the network.

This model is great for the client (and results in an architecture with many advantages): the client's only interface to CORBA is the set of functional operations it invokes on an object; no activation, deactivation or storage operations complicate its life. When it wants to make an invocation, it just goes ahead and does it. When it doesn't…it doesn't. If it wants, a client can hold onto an object reference for years without invoking it. Then, whenever it wants, it can send an invocation and expect that the object will be there to respond and will be in the same state in which it was left by the client's previous invocation – assuming that the object isn't shared with any other clients and hasn't been destroyed. Destruction of an object is irrevocable – once destroyed, it's totally gone and its object reference will never work again. In this article we'll discuss activation and deactivation, which are very different from creation and destruction. (If you're interested, we'll discuss creation and destruction of objects in a separate article.)

The model appears less efficient on the server side, however: Do we really have to keep all of our CORBA objects active, at least the persistent ones, on the off-chance that a client will send us an invocation at some random time? Suppose we have 10 million customers, each with his or her own shopping cart object, but only 2,000 of them are shopping now. We'd be overjoyed if the other 9 million-plus customers suddenly came back and started to shop all at once. But this is pretty unlikely, and we don't have enough computing power to run all 10 million carts at once anyhow. What can we do?

CORBA and its server-side mechanisms (known to the CORBA literati as the POA) deal with this situation by distinguishing the concept of CORBA object – the client's concept of an object that runs continuously from creation to destruction, maintaining its state all the while – from the concept of servant – a piece of running code that services an invocation. In the CORBA server-side model, object references are mapped to running code dynamically when needed – sometimes only when an invocation comes in, although there are other patterns as well. When an invocation completes, the resources it used can be freed to become available for whatever the server needs to do next. What's true is that an object is always available; what's not true is that it's always running.

This concept is necessary for scalability – the ability of a CORBA server to handle requests from millions of clients, using a reasonable number of computers. However, we've found that this concept is confusing or disconcerting to many when they hear about it for the first time. Here's a short story that illustrates, by analogy, the difference between object and servant, and demonstrates that it's perfectly possible for the client to cling to its concept of "CORBA object running all the time, waiting for me to invoke it" while the server allocates a servant for it only when an invocation needs to be serviced.

The Story

My younger son (who plays the role of the CORBA client in this story) believes in Santa Claus. Santa Claus is a fat and jolly old man who wears a red coat and pants trimmed with white fur, and travels in a sleigh pulled by eight or nine reindeer (depending on the weather), delivering toys to children around the world by jumping down chimneys and putting the toys into stockings hanging from the mantle. For a few days after Christmas, Santa recovers from this effort by soaking his feet in a hot bathtub and drinking hot chocolate. He spends the rest of the year making lists and collecting toys for the next Christmas. (This is the North American view of Santa Claus and the one we'll use in this story; if you're reading this article in another country with another view, please adopt our version for the sake of the analogy.

This article is based on the book CORBA 3 Fundamentals and Programming, *written and edited by Jon Siegel and published by John Wiley and Sons, Inc. © 1999 by Object Management Group.*

We know that in Finland, for example, Father Christmas walks through the front door and hands presents directly to the children. If Father Christmas is also a CORBA object, his encapsulation boundary is very different from the one we're about to describe for Santa Claus!)

The Santa Claus CORBA object that my son believes in supports a single interface with a single, well-defined operation, GetPresents. Here's the invocation:

- Wait until December 24, in the evening. Otherwise the invocation fails and returns the WrongNight exception.
- Find a stocking, preferably a very large one with your name on it, and hang it from the mantle above the fireplace.
- Fill a glass with milk, and set it on the hearth beneath the stocking.
- Cover a small plate with cookies and set it next to the glass of milk.
- Optionally, put a note requesting specific toys next to the milk and cookies. (The milk, cookies and note are input parameters.)
- Go upstairs to bed, and go to sleep.

Going to sleep is very important since it defines the encapsulation boundary, one of the key concepts in object orientation. Clients are not allowed to peek beyond this boundary: an object's interface is defined on it, and once the invocation has been delivered to it the invocation is in the realm of the implementation. In object orientation one reason for encapsulation is to enable substitutable implementations. Fortunately, encapsulation also enables scalability, as we'll see shortly. For the Santa Claus object, encapsulation is the only thing that allows the invocation to work at all.

But this invocation isn't going to work if we rely on the client's concept of CORBA Santa to fulfill it. Instead, we'll use the POA concept of servant – that is, some resource that gets activated and configured when needed, services a single invocation and is then released. Parents around the world will appreciate this next concept: in this story my wife and I play the role of servants. (Okay, at least in the POA sense!) Here's how the Santa invocation executes at our house:

- After son completes invocation, including going upstairs to bed, Mom and Dad stay up and wrap various non-Santa presents until we are sure that encapsulation requirements have been met.
- Mom fetches bag with various small gifts, and stashes them in stocking. New England tradition requires an orange in the toe, and family tradition requires a stuffed animal reaching out from the top (even though Dad thinks son is much too old for this).
- Dad takes the glass of milk, pours it back into the bottle, fills the glass up with eggnog and rum, grates fresh nutmeg onto the top and drinks it down.
- While drinking the eggnog, Dad and Mom may eat a cookie or two. (Mom doesn't like eggnog.) Most of the cookies get put back into the box. Part of one gets crumbled onto the plate, which remains on the hearth.
- Dad rinses the glass with milk to cover up the smell of the eggnog and rum, and places it back on the hearth next to the plate of cookie crumbs.
- Dad helps Mom put the last of the toys into the stocking and hang it back up on the mantle.
- Dad and Mom clean up the mess left over from present wrapping and all, and go to sleep.

In the morning son awakens and runs downstairs. Seeing the stocking full of toys and the empty glass and plate, he exclaims "Wow – look at all the toys. Santa Claus must be real – he left all these toys, and drank the milk and ate the cookies!" And from his point of view this is true: CORBA Santa accepted the input parameters (milk, cookies, optional note) and delivered the expected return value (toys). Son's point of view is undisturbed in spite of its conflicts with the reality of the implementation hidden beneath the encapsulation layer.

What Have We Learned?

Let's go over the CORBA lessons from this story.

1. Client and server can have totally different viewpoints of the object implementation, but as long as invocations get serviced according to the agreed-upon definition of the interface syntax and semantics, this inconsistency doesn't matter.

   In the story, son thinks that Santa is real and always exists, as we described at the start. In reality, the Santa Claus servant comes into being for only a few minutes a year – on Christmas Eve, when it's needed.

   In CORBA the client holds an object reference and acts as if the object always exists, making an invocation anytime it wants and assuming that the object will maintain its state from one invocation to the next regardless of the time that elapses between invocations. In reality, in POA-based systems, computing resources may not be allocated for an object until an invocation comes in and may be freed as soon as the invocation has completed. State is maintained on persistent storage between invocations, loaded on activation and stored again with any changes on completion.
2. There's a lesson here on scalability as well: the story about the jolly fat guy in the red suit may be charming, but as an implementation architecture it just doesn't scale. Too many kids need presents on the same night for any one person to distribute them all, especially a fat old guy who obviously doesn't keep in shape during his off-season, and the year between Christmases is too long for such a resource (even out of shape) to sit around unused. The POA-based implementation, however, doesn't have either of these problems: every household (well, almost) has a resource that can play the role of Santa Servant on one night a year, so there's no problem scaling to any number of households. And the resource is flexible enough to play other servant roles during the rest of the year, whenever the household POA requires it.

The object-oriented principle that enables this is encapsulation: the implementation is encapsulated beneath a boundary that the client is not allowed to penetrate. It's been common to suggest that implementation details such as algorithm and coding reside on the far side of this boundary and may change unbeknownst to the client. With the POA, CORBA now allows a resource allocation infrastructure – which may be massive, supporting a huge enterprise application or a worldwide e-commerce shopping site – to lurk beneath this boundary.

The CORBA Component Model, the newest piece of CORBA 3, defines a standard server-side architecture based on the POA with an easier-to-program environment that's integrated with Enterprise JavaBeans. It has all of the architectural advantages we described in the foregoing short story.

Where to Learn More

The CORBA standard is written and maintained by the members of the nonprofit Object Management Group (OMG); all of the group's standards are available from OMG's Web site (www.omg.org) free of charge. Products implementing the standard are available from more than 70 sources ranging from most of the best-known companies in the computer industry to small independents, R&D labs and academic institutions. For more information visit the Web site at the URL above, pick up a copy of my book, *CORBA 3 Fundamentals and Programming*, or send an e-mail to info@omg.org.

siegel@omg.org

AUTHOR BIO

Jon Siegel, Object Management Group's director of technology transfer, presents tutorials, seminars and company briefings around the world. He has extensive experience in distributed computing, OO software development and geophysical computing. Jon holds a Ph.D. in theoretical physical chemistry from Boston University.

SYS-CON Radio Interview

GEORGE KASSABGI VICE PRESIDENT OF MARKETING

APPTIVITY PRODUCT UNIT, PROGRESS SOFTWARE

JDJ: *Tell us a little bit about Progress Software, and then move into the Java Internet products if you would.*

Kassabgi: Progress Software is a large software company. We had revenues last year of $280 million as a publicly traded company. Basically, we sell the number one embedded database, the Progress Database. We have consistently been a favorite among the value-added reseller community, people that basically sell packaged applications built on top of our technology, which as an ensemble sold some $1.5 billion worth of software packages last year alone. The Apptivity product comes from the Apptivity Product Unit of Progress Software, which has been in place since early 1997.

Apptivity 3.1, released in July, provides significant movement, continued movement, into the world of open platform application servers, the productive environment for the development and deployment of business applications using the Web and open standards, which include Java, CORBA, EJB, IIOP and so on.

SonicMQ, to be released later this month, provides a Java Messaging Server that allows developers to create distributed applications using the JMS standard. The SonicMQ product provides such valuable features as publish/subscribe, point-to-point communication, message transactions and queuing.

JDJ: *You said a magic word there. A lot of people were talking about application servers at JavaOne. A ton of them are out there. Somebody asked how many application servers he had to have running on his server to do everything he wants. What sets Apptivity apart from other application servers?*

Kassabgi: Clearly this is a tremendous following for the EJB standard, which is what many of the application servers tend to focus on. So, yes, there are many AP servers. The clear differentiation is performance, number one; number two is the feature functionality, the robustness and productivity with which feature functionality has been put forward; and three – I think a differentiator – is the kinds of customers and reference accounts and applications that have been deployed with a given application server product. A prospect should look at all three of those things. I'd actually add a fourth that is very important: the type of vendor putting the application server forward – is it a start-up company whose future is oftentimes unknown? Is it a platform vendor – for example, a large hardware vendor? That kind of company may have a unique style in putting forward, let's say, a softer product. Or is it a vendor like Progress Software that for the past 15-plus years has been doing nothing but providing productive environments for developers?

JDJ: *What sets SonicMQ apart from other messaging servers?*

Kassabgi: SonicMQ is the first JMS server available from an established software vendor. We see other vendors shipping JMS implementations over the next 6–9 months and look forward to competing against them along the lines of product features, performance and scalability. In addition to these competitive stances, we also believe we have the best business acumen for working with ISVs and ASPs who seek to embed the JMS technology within their systems.

JDJ: *You mentioned performance. What are some of the characteristics of Apptivity and SonicMQ performance?*

Kassabgi: I think it's essential because if you have an AP server and it doesn't scale, it doesn't perform, then you don't have a product that has inherent value beyond a demo or an initial review by the developer. I'll let the unbiased third-party DocuLabs benchmark speak for itself. We were far and away the best-performing AP server in the list of AP servers at JavaOne – you know, the AP servers that used the Java open platform standard.…The next best AP server behind us was less than half the transactions per minute.

With SonicMQ we intend to outperform all other JMS implementations as they appear on the market. Of great importance is the benchmark of messages per second, particularly under heavy load and with transactional or guaranteed messaging requirements.

JDJ: *What kind of developers are out there using Apptivity and in what way?*

Kassabgi: Developers, both IT teams and people, ISVs, that are building applications, and then people in general who have standardized on the Java platform and want to build NT or business applications with significant business logic on the server with Enterprise JavaBeans as the business logic components. You have the flexibility of both HTML client type as well as Java client type, the use of XML as the data exchange between functions and between servers. You have developers that have made those kinds of choices and have standardized on Java and want to build an application for internal use or for resale and want to be able to rely on a vendor like Progress Software to provide that kind of infrastructure over time. Clearly, those developers are looking for something that's proven to perform and scale because the application is going to have to handle a large number of users and a large level of throughput once it's deployed.

The secret is understanding what makes the application successful at deployment. While what a developer wants at development time may be evident, what's far more important is what the vendor can actually provide in facilitating the deployment and making it successful. And I think the benchmarks show a very real glimpse of what happens at deployment and what kind of performance one can expect to have.

JDJ: *Can you be a little more specific about who some of those customers are, and something that our [readers] might be able to go check out?*

Kassabgi: Some of these are prominent on our Web site with some success stories. The number one Web portal in the world, Yahoo!, uses Apptivity. They use an Apptivity application to manage all their Web advertising on the portal. That is a success story that the listeners can look at on the Web site Apptivity.com. Also, Chicago Mercantile Exchange, which is basically a who's-who in commodities trading, uses Apptivity to do commodity trading among its member firms. Futures Online in Chicago has created a similar kind of system with Apptivity. AT&T Calling Card Division in Jacksonville, Florida, uses Apptivity to process calling card transactions. That's a short list of very notable customers. [People], again, can go to the Web site and find out more about these. Keep in mind that to have a reference account means it must be deployed and it must be successful.

JDJ: *What is the vision for the products moving forward?*

Kassabgi: That's an important part. The vendor needs to provide the best possible product today, and then the vision, moving forward, so that its customers can be carried along and get the benefits from that. The vision of the product is to move further into the capabilities of the Enterprise JavaBean specification and to do so in a manner that allows us to continue having the scalability and performance that can lead the market. It's one thing to say that you'd support an element of the EJB specification for the sake of supporting it. It's another to say that you'd support it in a manner that allows you to basically perform and scale to the highest degree possible. It's our desire therefore to continue going further in the implementation of the Enterprise JavaBean specification along those lines.

In addition to that, we regard Java messaging service, JMS, as extremely important for distributor applications and for applications where data is being dealt with in an intercompany or intracompany fashion, messages basically being the lingua franca of many of tomorrow's Web

applications. So we're adding full JMS capabilities to the product in the not too distant future as well as continuing to enhance the ability to use XML as the actual language for the data exchange that would happen via messaging and then, in addition to all that, continuing to work to make the product not only very productive in its current state – namely, an integrated application environment state – but also make it very productive for developers that would be using other IDEs or tools with Java. And so, continued emphasis on productivity as well.

“The number one **Web portal** in the world, **Yahoo!**, uses **Apptivity.** They use an Apptivity application to **manage** all their **Web Advertising** on the portal. That is a **success story** that the **listeners can look at** on the Web”

We see the vision as being something that tackles the demanding requirements of the developer that has chosen open platforms and the Java platform, demands working scalable performance implementations of EJB and JMS, demands that XML be threaded throughout the product in a cohesive manner, be able to use XML as the data exchange throughout and wants all of that in a concise package where there are few restrictions, few limitations, and where deployment results in a successful deployed business application with potentially hundreds – thousands – of end users and tens of thousands of transactions per second.

***JDJ:* If I want my company to have their Web site perform with transactions at a speed and consistency that a Yahoo would but I can't afford it, then I'm out of the picture. Can you tell us how the software is bundled and what the price correlation is?**

Kassabgi: We've had the same pricing model for quite some time. It's not changing as a result of 3.1. The pricing model is very straightforward, and I think you'll find most if not all AP servers are very similar, taking a similar stance, which is the developer copies are $995 per developer seat. That allows the developer to develop an application and also to test-deploy it on his or her system.

Then there's a deployment pricing – when you deploy a business application built with Apptivity there's a deployment license. The prices start at $10,000 per CPU utilized in the deployment. That, again, is kind of the value-to-cost proposition there. The ratio is consistent with most or all other AP servers or better than other AP servers, but it's basically a deployment CPU-based licensing model. SonicMQ, our JMS product, is priced at $3,000 per CPU utilized in deployment and the developer copy is made available for free over the Web on www.SonicMQ.com.

***JDJ:* You mentioned the Web site before. Can you tell us what we can find there?**

Kassabgi: What you'll find at www.Apptivity.com and www.SonicMQ.com is an exposé and a number of the reference accounts – success stories, as we call them – with a fair amount of detail. There's even an extensive look at some applications that have been built and how they were built, and examples of the kind of thought process behind it. I believe the example that's on there is an application used by Scholastic Inc., the number one distributor of children's books in New York City. That application is an example of where Apptivity was used to integrate data from Legacy application on PeopleSoft human resource. You'll also find a detailed explanation and white papers on what the product is all about and what it does, and the ability to download the product, the developer license. Developers can download that and…evaluate it, and there's no better way to learn about a product than to use it. I think there's also a link of all the press releases and notable mentions, the so-called “In the News” section.

Java for Linux

Interest is growing in these two technologies

WRITTEN BY
HARRY FOXWELL

Two technologies that have gained widespread interest and support in the past few years are Java and Linux. Until recently, however, the two were separate, although they share similar visions of open, ubiquitous computing. As interest in both Java and Linux solutions increases, developers are naturally looking to combine the two, and want to write applications in Java that run on Linux systems. But they need stable, fast and fully functional JVMs, especially for server solutions that require servlets and support for Java Server Pages.

Several independent efforts to port JVMs to Linux are in progress, including those by IBM, Sun Microsystems, the Blackdown Java-Linux Porting Team headed by Steve Byrne and the Kaffe group headed by Tim Wilkinson. The IBM and Blackdown ports are based on code licensed from Sun, while Kaffe is a “cleanroom” implementation. Ports based on Sun code are complicated – and frequently delayed – by licensing issues. While IBM has an industry license for Java technologies, the Blackdown group had to make special licensing arrangements to obtain the Java 2 JVM source code. Sun’s Community Source Code License may eventually make it easier to resolve these licensing issues, although the prevailing opinion within the Linux community seems to favor the GNU Public License.

Current JVMs for Linux

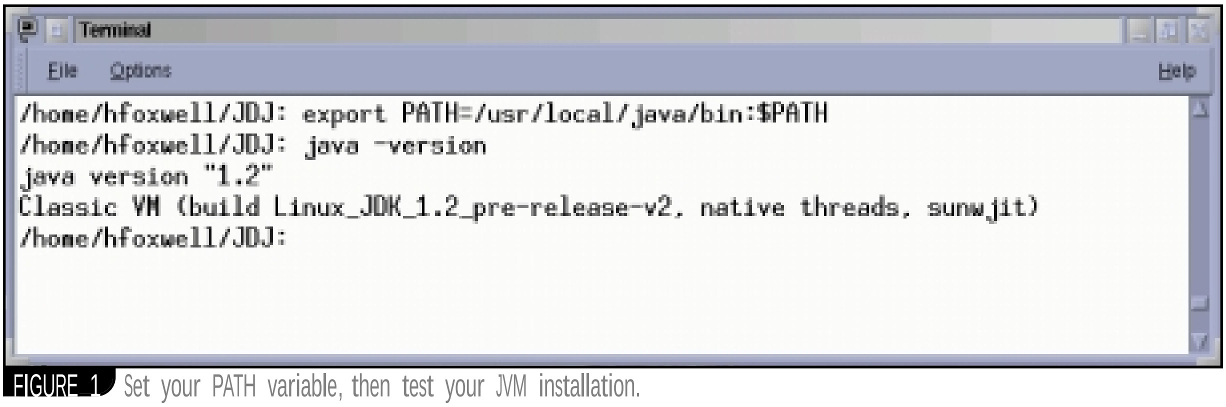
Sun currently supports reference implementations of JDK 1.1 and JDK 1.2 (now called Java 2 SDK) for Solaris and for Windows 95 and NT. Also available are highly optimized production versions for Solaris and Windows. At the JavaOne Developer’s Conference this past June, Sun announced a restructuring of the Java APIs and specifications into the Java 2 Micro Edition for handheld and embedded applications, the Java 2 Standard Edition for workstation applications and the Java 2 Enterprise Edition for server applications. The Micro Edition, for example, excludes some libraries – such as the AWT classes – that aren’t needed for embedded systems and handheld devices. The Enterprise Edition targets server application development and supports Enterprise JavaBeans classes. The Java for Linux efforts now in progress target the Standard Edition, which includes the Java Foundation Classes.

Available Linux JVM ports are all based on the “Classic” JVM reference code and don’t include any of Sun’s HotSpot or other performance-enhancement code.

IBM’s port is an early-access release of the 1.1.8 JVM, and includes a JIT compiler and support for native threads. Linux users who have worked with this JVM report good performance and stability. IBM is reportedly working on a Java 2 port, but they correctly observe that there is still much demand for 1.1 applications and that 1.2 isn’t widely used yet.

The Kaffe JVM, included with some Linux distributions, is a cleanroom implementation of the 1.1 JVM specification. But the most recent release, 1.0b4, doesn’t include RMI support and is missing several crucial security features such as bytecode verification and observance of “private” modifiers. Tim Wilkinson, who leads the Kaffe effort, says several improvements are in progress, including a significantly faster JIT compiler and support for the Java 2 Collections API.

The Blackdown ports of the 1.1 and 1.2 JDKs are probably the best-known implementations for Linux. Led by Steve Byrne, a former Sun employee, the team has been working on a fully JCK-compliant 1.2 JVM for nearly a year. The current “prerelease” is fairly complete and stable, and includes a JIT compiler.



```
/home/hfoxwell/JDJ: export PATH=/usr/local/java/bin:$PATH
/home/hfoxwell/JDJ: java -version
java version "1.2"
Classic VM (build Linux_JDK_1.2_pre-release-v2, native threads, sunwjit)
/home/hfoxwell/JDJ:
```

FIGURE 1 Set your PATH variable, then test your JVM installation.

Sun has provided some engineering support to the Blackdown group for JDK ports, and has even worked with several Linux distributors such as RedHat to help port Linux itself to UltraSPARC systems. Linux developers would like to see Sun provide a supported reference implementation of the Java 2 SDK, but Sun has made no announcements about its plans for supporting Java on Linux.

Java Programming on Linux Systems

You can download Java Developer Kits for Linux from IBM (www.ibm.com/java) and Blackdown (www.blackdown.org). Installation consists of uncompressing the downloaded file and extracting the component files and directories. Linux includes utilities for uncompressing ZIP files and other compression formats. The Blackdown distribution file jdk1.2pre-v2.tar.bz2 is compressed with the bzip2 program. To extract and install the files, use these two commands:

```
bzip2 -d jdk1.2pre-v2.tar.bz2
tar xvf jdk1.2pre-v2.tar
```

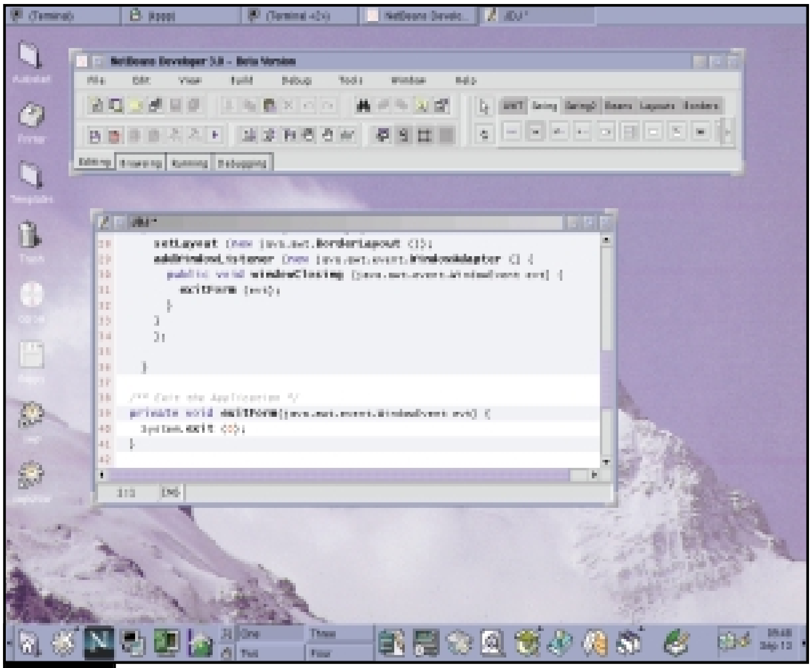
After you install the files in a directory, you need to set your PATH environment variable so your command shell can find the javac compiler, the java runtime program and other Java utility programs. Also, if you plan to use third-party Java libraries, you must set your CLASSPATH environment variable to include the directory location of those libraries. For example, I set my Java environment variables for my command shell, ksh, like this:

```
export PATH=/usr/local/java/bin:$PATH
export CLASSPATH=/home/hfoxwell/MyJava
```

Test your installation by entering "java - version" from your command line. The JVM should start up and report "java version 1.2" or something similar. If you're testing more than one JDK, be sure to set your PATH and CLASSPATH explicitly for each version that you plan to use (see Figure 1).

Linux JVMs typically require a "glibc" C library in order to run. Check the Release Notes for your JVM to determine which version of this library is required. Most current Linux distributions ship with one or more versions of "glibc". The Blackdown 1.2 JDK requires "glibc2", also called "libc6", and is found at "/lib/libc.so.6".

All of the current Linux JVM offerings are unsupported beta or early-access releases. As such, they are incomplete, buggy and not fully optimized for performance. The Blackdown 1.2 JVM doesn't have a working native threads module yet, and the IBM port is missing some IO exceptions. Moreover, important extension APIs, such as Java 3D, Advanced Imaging, and Sound, aren't yet ported to Linux. Sun has provided source code for these to IBM, Blackdown and other licensees, but it may be a while before you see these APIs on Linux.



```
setLayout (new java.awt.BorderLayout ());
addWindowListener (new java.awt.event.WindowAdapter () {
  public void windowClosing (java.awt.event.WindowEvent evt) {
    exitForm (evt);
  }
}
);

}

/** Exit the Application */
private void exitForm(java.awt.event.WindowEvent evt) {
  System.exit (0);
}
```

FIGURE 2 NetBeans provides an easy-to-use Java IDE for Linux.

Java Development Tools for Linux

At JavaOne several vendors announced plans to provide developer tools for Linux, including IBM, Inprise and MetroWerks. Blackdown's Web site includes an extensive "Java Tools for Linux" link that lists IDEs and other programs for Java developers. If you're familiar with "Emacs", have a look at "emacs-JDE" at www.sunsite.auc.dk/jde/. If you'd like a full-featured, drag-and-drop IDE, try NetBeans from www.netbeans.com/ (see Figure 2). NetBeans is written in Java and runs nicely on Linux using the Blackdown 1.2 JVM. It supports development in Java 1.1 or 1.2, and is distributed in a free personal-use version as well as a commercial, enterprise development version.

Conclusion

Open source advocates have raised the awareness of Linux as a stable, richly featured operating system. They're also attracted to Java's platform-independent programming model and to the new solutions it enables, such as Jini, Enterprise JavaBeans and JSPs. As Linux JVMs from Sun, Blackdown, IBM and others become more stable, complete and competitive, Linux developers will be able to contribute their considerable expertise to the widespread deployment of Java technologies.

URL Resources

1. GNU Public License: www.fsf.org/copyleft/gpl.html
2. IBM's Java for Linux: www.alphaworks.ibm.com/tech/linuxjvm
3. Blackdown Java-Linux Porting Team: www.blackdown.org/
4. Netbeans Java IDE: www.netbeans.com/
5. Kaffe: www.kaffe.org/
6. Metrowerks CodeWarrior for Linux: www.metrowerks.com/desktop/linux/
7. Sun's Linux information site: www.sun.com/linux

harry.foxwell@east.sun.com

AUTHOR BIO

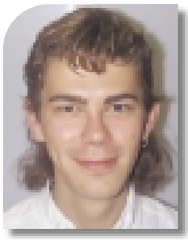
Harry Foxwell is a systems engineer for Sun Microsystems' Southern Area, responsible for technical consulting and education on Java and network computing. Harry has worked with Java since its introduction in 1995, and is a contributing author to Introduction to Programming Java Applets, Java Core Libraries and Java for Managers, instructional CDs published by MindQ. He also maintains Sun's internal Web site of Linux information.

Sun Acquires NetBeans, Supplier of Java-Based IDE for LINUX

In October Sun Microsystems, Inc., announced the purchase of NetBeans, developer of cross-platform Java-based integrated development environments. In conjunction with products from Sun's simultaneous acquisition of Forté Software, Netbeans IDE products will provide JDK software customers with software choices ranging from entry-level offerings for individual developers and students to enterprise-class offerings for team development. Free versions of NetBeans Developer IDE for Solaris, Linux and Windows 98/NT can be downloaded from www.netbeans.com/product_dl.html.

Software Development **Productivity**

SoftWired optimizes Java messaging for the enterprise using JProbe



WRITTEN BY
SAM WATTS

The distribution of business intelligence through a network of organizations within an enterprise requires the evolution of spontaneous networks, which in turn requires middleware that facilitates intelligent communication of information regardless of platform, device or application. In this context Java provides a natural platform-neutral object-oriented approach that supports a scalable messaging service, JMS (Java Messaging Service), now a standard part of the Java 2 Enterprise Edition (J2EE).

Leading the way with a JMS interface to their messaging service is SoftWired, whose iBus//MessageBus is the first of seven products that allow companies to build scalable electronic business systems more easily. iBus distributes messages and business events through various communication protocols to diverse applications. SoftWired decided to use Java for this product as early as 1996, when few tools were available to assist developers with this relatively new language.

Lightweight messaging middleware written entirely in Java is the core of SoftWired's iBus solution. This publish/subscribe middleware allows any type of computing device (PC, server, mainframe, PDA, cellular phone) to efficiently exchange information by any communication protocol (IP multicast, TCP/IP, HTTP, wireless).

Working as a unit of nine developers, SoftWired needed to ensure that MessageBus was fast and efficient enough to deliver business events in near real-time. When Silvano Meffeis, executive vice president of SoftWired, encountered a performance problem in the module that set up the network configuration, he used JProbe Profiler from KL Group to find the cause.

He had suspected that the problem lay in serialization/deserialization of the network objects. "That was only a guess until I ran Profiler, which pinpointed exactly where we needed to optimize." JProbe soon became an indispensable part of SoftWired's development: "JProbe Profiler enabled us to enhance the performance of our Java messaging middleware by 50%."

Meffeis also found performance bottlenecks in unexpected places. He found that using the "+" operator to concatenate strings was simple to code but expensive to run, since each concatenation actually created a StringBuffer object that was thrown away after use. Detecting performance problems related to string manipulation would have been difficult without tools such as Profiler, since the calls to these methods were distributed throughout the package. After witnessing the effects of the methods on the system's overall performance in Profiler, Meffeis decided to use them less often. By converting the strings to byte arrays before transmission and converting them back to strings only if required, Meffeis improved the performance of the module by 20%.

Using JProbe Profiler on MessageBus was just the beginning. JProbe's Memory Debugger (fully integrated with JProbe Profiler) enabled SoftWired engineers to find message objects that were no longer being used but still taking up precious memory. Once these objects were located, following the stack trace back to where these objects were created was easy, enabling the developers to manage them more effectively.

Senior software engineer Bill Kelly of SoftWired used JProbe Threadalyzer to look for race conditions or missing locks on data. "I ran some tests and it pointed out how a few messages should have been synchronized, things I'd never have found myself."

Threadalyzer also caught a race condition. Race conditions and other thread problems manifest themselves rarely, but when they occur, the results can be unexpected and catastrophic. "We hadn't noticed any symptoms," Kelly says. "We were lucky." Debugging thread

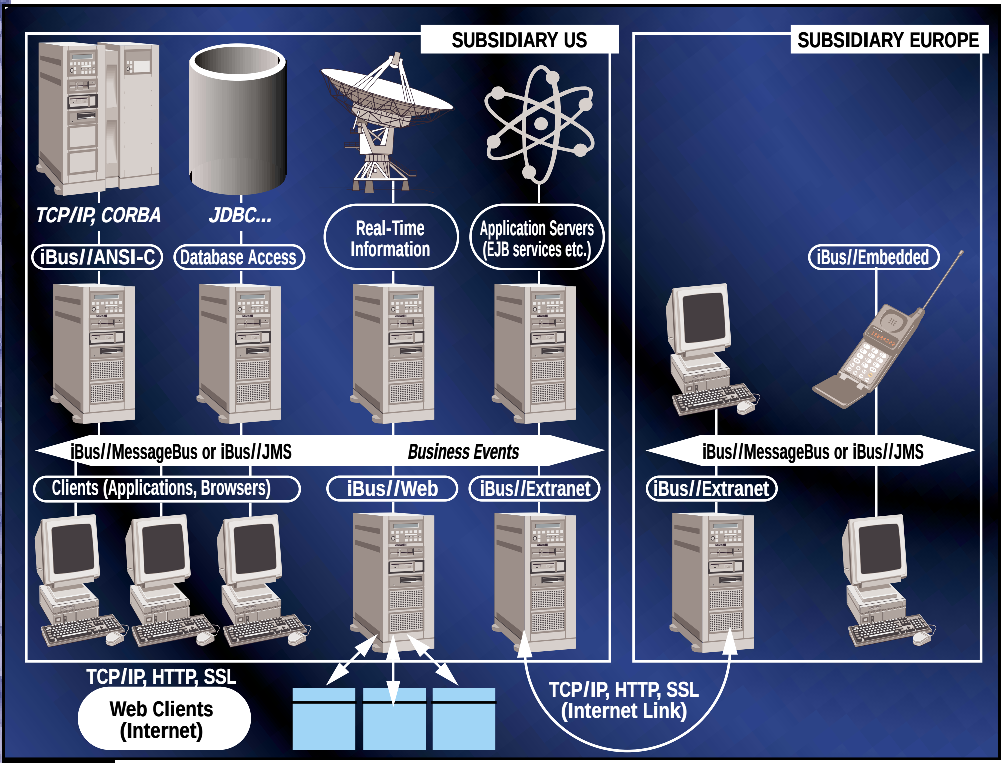


FIGURE 1 SoftWired's messaging solution for e-business

problems without tools is notoriously difficult. “If [the race condition] had ever come up in practice, it would have been extremely complicated to try to find.”

Kelly says, “If you’re working on software that’s already complex, the threading issues can have devastating consequences. Having performance-tuning and thread analysis tools to look over your shoulder is essential.” He also found JProbe Coverage useful for establishing which parts of the system had not been executed and required testing. “Having a coverage tool can motivate you to test better. It’s fun to see yourself getting closer to the coverage level you’ve set your sights on.”

Working out the problems of the iBus//MessageBus modules early was important to SoftWired. Releasing iBus//JMS in September 1999 was a key milestone. With JMS a standard part of J2EE, application servers will need to upgrade to this level of service, and at present few application servers have a JMS implementation.

SoftWired’s JMS interface means that their iBus solution can be combined with an application server and SoftWired add-on products to constitute a complete set of J2EE-compliant services. With an implementation in 100% Pure Java, accurately profiled and tested for performance, memory, and thread and test coverage problems, SoftWired’s engineers are confident that these products form the ideal building blocks of large and complex networks, spanning platforms, systems and protocols.

Author Bio

Sam Watts is studying computer science at the University of Waterloo, Ontario. He specializes in Java development.

sgwwatts@undergrad.math.uwaterloo.ca

JProbe 2.5 ServerSide Edition Now Available

Server-side Java demands high performance and reliable code. When a server is providing output to thousands of users, it can’t afford to run into performance bottlenecks, memory leaks or threading problems. JProbe 2.5 ServerSide Edition brings powerful server-side performance-tuning capabilities, full Solaris support and IBM VisualAge integration to this award-winning suite. The product includes JProbe Profiler (with integrated JProbe Memory Debugger), JProbe Threadalyzer and JProbe Coverage.

Server Launch Pad

The new JProbe Server Launch Pad enables easy point-and-click integration with major Web and application servers, such as:

- IBM WebSphere
- BEA WebLogic
- Java Web Server
- Allaire JRun
- Sun Servletrunner

Solaris

JProbe ServerSide Edition also introduces suite-wide support for Solaris 2.6 and 7 for developers working in JDK 1.1 or Java 2 (JDK 1.2).

IBM VisualAge

IBM VisualAge for Java stores all Java files in its own source code repository, which can make tuning an application a lengthy and complex process. With JProbe 2.5 support for VisualAge, tuning an application, EJB or servlet written in VisualAge for Java is quick and simple.

JProbe Threadalyzer Enhancements

JProbe 2.5 (both Developer and ServerSide Editions) introduces new lock analyzers that extend the predictive value of JProbe Threadalyzer, and an all-new Visualizer that provides a more graphically intuitive means of detecting thread problems.

Associations for EJBs

Representation and support for associations can be a crucial issue when implementing object-oriented systems

WRITTEN BY
SCOTT DANFORTH

Object-oriented systems analysis and design typically yield an object model whose classes are organized using inheritance and associations. *Inheritance* represents common interfaces and behavior among different classes of objects. *Associations* represent statically typed binary relationships between objects that are established and modified during operational use of the system.

When the object model for a system is implemented using an OO programming language, classes and inheritance relationships of the model can be mapped directly into statically defined classes within the language. But associations between objects aren't supported as OO programming language primitives. Thus representation and support for associations can be a crucial issue when implementing object-oriented systems.

This article considers how associations might be represented and supported within the framework offered by Enterprise JavaBeans.

Important Aspects of Associations

ASSOCIATIONS ARE DETERMINED BY ROLES

Due to the binary nature of its corresponding relationship, an association is determined by two roles whose names and static types characterize the objects that participate in the relationship. For example, if an association has two roles of type Person, named mother and child, a Person object that plays the mother role is expected to be (in a representational sense) the mother of the related Person objects playing the child role. A UML diagram representing such an association might appear within Rational Rose as shown in Figure 1.

CARDINALITY AND NAVIGABILITY DESCRIBE ROLES

Roles have an associated cardinality. The above model specifies that a child has exactly one mother, but a mother can have any number of children. The other important aspect of a role is navigability. In Figure 1, for purposes of illustration, a directed association link is used to model a system in which a mother knows her children, but the children don't know their mother.

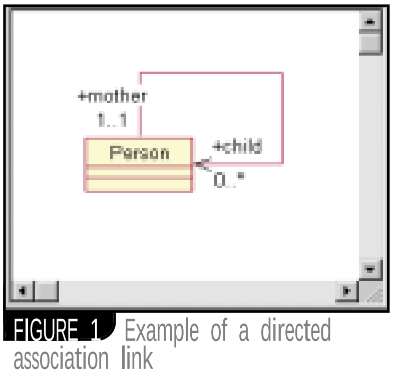


FIGURE 1 Example of a directed association link

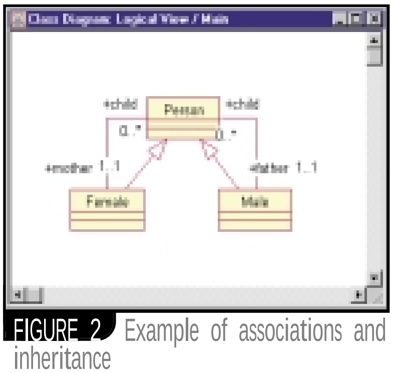


FIGURE 2 Example of associations and inheritance

ASSOCIATIONS AND INHERITANCE ARE INTEGRATED BY AN OBJECT MODEL

Associations and inheritance appear in an integrated manner within an object model. Figure 2, derived by elaborating on the initial example in Figure 1, illustrates this by presenting a model in which every Person has exactly one mother and one father, and every Female and Male is a Person and may have any number of associated children.

ASSOCIATIONS CAN BE REPRESENTED BY METHODS

When a class of objects is modeled as participating in an association and the related role is navigable, instances of the modeled class are intended to have some way of accessing the objects that play that role. It's therefore reasonable to postulate the existence of a method for computing or accessing an association "role extent" for instances of the class.

As there are four navigable roles in the example above, four different methods would be provided to access role extents. A vector can be used to return *n*-ary role extents.

```
public Female Person.mother();
public Male Person.father();
public Vector Female.children();
public Vector Male.children();
```

In keeping with the use of associations to represent dynamically changing relationships, association modification methods are often appropriate. Depending on the system being implemented, a class that participates in associations may provide "add" and "remove" methods corresponding to its associations' opposing roles. In the current example these methods could appear as follows:

```
public void Person.addFather(Male f);
public void Person.removeFather(Male m);
public void Person.addMother(Female m);
public void Person.removeMother(Female m);

public void Female.addChild(Person c);
public void Female.removeChild(Person c);

public void Male.addChild(Person c);
public void Male.removeChild(Person c);
```

An important aspect of association modification is that it generally affects role extents for multiple objects on both ends of an association. For example, adding or removing an association between a mother and child will affect the extent of the child role (accessed from the mother) as well as that of the mother role (accessed from the child).

Thus, implementing the dynamics of an association requires more than supporting references between objects.

The approach illustrated above provides a simple and intuitive basis for supporting associations from within an object-oriented programming language: object methods are used to maintain associations and access navigable role extents. But how are these methods to be implemented? The following section offers an approach that might be appropriate for container-managed Enterprise JavaBeans.

EJB Developer vs Deployment Responsibilities

We're concerned here with how to support persistent associations between container-managed entity beans.

DEVELOPER RESPONSIBILITIES

1. The EJB developer determines an object model and provides EJB classes corresponding to this model. The EJB classes provide various "association methods," as illustrated above, that are used by clients (through remote interfaces) and by other EJBs to maintain associations and access navigable roles.
2. The EJB developer uses container APIs (from an extension of the EntityContext interface) to implement the association methods.
3. The EJB developer communicates essential aspects of the object model to the deployment phase within the deployment descriptor to allow the container to support the required associations via its APIs.

DEPLOYMENT RESPONSIBILITIES

1. The EJB deployer inspects the object model provided by the deployment descriptor and implements the required object model associations in a manner appropriate for the specific EJB container technology being employed. For example, when using EJB containers based on object-relational mappings, association tables might be determined and created (in addition to any database tables required for storing object state). For containers based on single-level store object technology, other implementations might be appropriate.
2. The resulting database schema and other implementation information for the model associations are provided to the container in whatever way is appropriate for the particular container being used. This information is what enables the container to support object-model associations through the association-related APIs in the container's extended EntityContext interface.

AN EXTENDED ENTITYCONTEXT INTERFACE WITH ASSOCIATION SUPPORT

An extended container interface provides complete support for computation and modification of role extents. In the simple approach suggested in Listing 1, a vector is used to hold role extents that, after registration, are kept up to date by the container when it executes association modification operations.

USING ASSOCIATIONSUPPORT

EJBs use the extended container interface to compute and modify role extents. When a role extent is first needed, an EJB class creates an appropriate collection for the extent and then registers it with the container. After this, the container maintains the role extent as required when add- or remove-association methods are called. Listing 2 illustrates how this appears in the case of the Person class.

COMMENTS ON THE CODE

The most important aspect of the approach in Listing 2 is that it relies on the container to compute and maintain role extents. This corresponds with other container responsibilities in the case of container-managed entity beans. In general (assuming bidirectional navigability), two role extents will be updated by the container when an association is modified. Of course, the container will also modify whatever container-dependent state is used to support persistence and the initial registration API. For example, in the case of object-relational mappings, a persistent association table containing foreign keys of associated objects might be updated.

Implicit in Listing 2 is the idea that an EJB is allowed to access other EJBs (not their remote interfaces) directly via registered role extents. Direct access to another EJB within the same container is actually a novel idea in the context of the EJB specification, so the act of registration seems appropriate. For one thing, a container needs to know which objects can and cannot be passivated.

Listing 1

```
public interface AssociationSupport extends
javax.ejb.EntityContext
{
  public void registerRoleExtent(Object target,
                                 String roleName,
                                 Vector roleExtent);
  public void addAssociation(Object target,
                             String roleName,
                             Object rolePlayer);
  public void removeAssociation(Object target,
                                String roleName,
                                Object rolePlayer);
}
```

Listing 2

```
     public class Person {
         AssociationSupport ctx;

          public void setEntityContext(EntityContext _ctx) {
                  ctx = (AssociationSupport) _ctx;
      }

         public Male father() {
                  if (fatherExtent == null) {
                           fatherExtent = new Vector();

ctx.registerRoleExtent(this, "father", fatherExtent);
                  }
                  return (Male) fatherExtent.elementAt(0);
         }
         Vector fatherExtent = null;

         public void addFather(Male f) {
                  ctx.addAssociation(   this, "father", f);
         }
         public void removeFather(Male f) {
                  ctx.removeAssociation(this, "father", f);
         }

         public Female mother() {
                  if (motherExtent == null) {
                          motherExtent = new Vector();

ctx.registerMotherExtent(this, "mother", motherExtent);
                  }
                  return (Female) tmp.elementAt(0);
       };
       Vector motherExtent = null;

       public void addMother(Female m) {
                  ctx.addAssociation(this, "mother", m);
       };
       public void removeMother(Female m) {
                  ctx.removeAssociation(this, "mother", m);
       }
   }

   ... similarly for Male and Female
```

Author Bio

Scott Danforth works at Secant Technologies, Inc., in Cleveland, where he focuses on the development and practical application of object-oriented technology. He has a Ph.D. in computer science from the University of North Carolina at Chapel Hill, and is the coauthor of Objects for OS/2 *and* Putting Metaclasses to Work.

Listing 2 assumes a single global name space for role names and that the class of a target object and a role name uniquely determine an association. A more realistic approach might be to scope role names within association names. Association names for the above example might be chosen as M and F, resulting in the fully scoped role names M.mother, M.child, F.father and F.child.

OTHER ISSUES

We haven't discussed the client API for associations. Vector provides a simple collection class that's appropriate for the general-purpose EJB/container API for associations. But the client API for maintaining and accessing specific association roles (provided from the remote interface of the target object) can use types specific to the particular association role involved. For example, when accessing *n*-ary role extents containing a particular type of object, the collection interface used for finder operations on that type would seem appropriate.

> "Inheritance and associations are object modeling ideas that are important when implementing software for complex, real-world systems"

We've avoided discussing container-specific mechanisms made available to deployers in support of associations. However, the overall approach suggested above is consistent with that used by Secant Extreme Enterprise Server for EJB. There, deployment includes creation of association tables and generation of a "metadata" file that describes these tables, the associations they support and the corresponding EJB classes. The metadata file is provided at runtime to a Secant EJB service (container) that loads the corresponding EJB classes and then uses the indicated tables to support persistent associations.

LOCAL/REMOTE ACCESS

In Listings 1 and 2 the expectation is that associations between EJBs are made visible to remote clients through the beans' remote interfaces. However, "local" associations (available to support business logic but not visible to remote clients) might be useful in some cases.

Conversely, it seems reasonable that an application assembler might want to associate EJBs from separately developed object models. In this case the remote interfaces of the selected EJBs would be provided with association methods, and deployment would provide their implementations (instead of deferring to EJB classes). In this case client applications would be able to make use of associations, but business logic would not.

Thus we suggest "local" and "remote-only" as special kinds of associations in the deployment descriptor. Local associations would be specified by a bean developer and supported by EJB classes. Remote-only associations would be specified by an application assembler and completely supported by remote interface objects. Both possibilities seem potentially useful.

Summary

Inheritance and associations are object modeling ideas that are important when implementing software for complex, real-world systems. The purpose of this article was to condense these general ideas into a simple, concrete form that I hope will be useful for supporting discussion and further work in this area.

scott@secant.com

PRODUCT REVIEW

VantagePoint 4.0

by Visualize, Inc.

Data visualization class library worth the effort

REVIEWED BY JIM MILBERY

AUTHOR BIO

Jim Milbery is a software consultant with Kuromaku Partners LLC, based in Easton, Pennsylvania . He has over 15 years of experience in application development and relational databases. Jim can be reached via the company Web site at www.kuromaku.com.

jmilbery@kuromaku.com

Test Environment
Client/Server:
Dell Precision 410,
128MB RAM,
10 gigabyte disk drive,
Windows NT 4.0 (Service Pack 4)

Visualize, Inc.
1819 East Morten, Suite 210
Phoenix, AZ 85020
Phone: 602 861-0999
www.visualizeinc.com

Founded in 1996, Visualize, Inc. develops and markets a series of Java-based products for interactive data analysis and visualization. They announced the availability of the latest version of their VantagePoint data visualization class library at JavaOne back in June, and I was recently given the opportunity to test out the new version of the software.

Understanding VIDA

Visualize recommends that you have some experience working with Java programs and data visualization before you start working with VantagePoint. The key to your success with the product is understanding Visualize's VIDA (Visual Interactive Data Analysis) architecture. VIDA is the process by which numerical data is translated into pictures that are more intuitive for end users to understand. VantagePoint supports a wide variety of these 2D, 2½D and 3D visualization objects (charts and graphs) with hooks that allow you to add real-time drill-downs. Visualize provides precooked example programs that show you just how this interaction process works, and I found these examples incredibly helpful in mastering the product.

Installation and Configuration

Visualize makes the software available through their Web site. The VantagePoint download is delivered as an InstallShield Java Edition program, which means you'll need to have a copy of the JDK installed in order to run the installation program. Installation is quick and painless and the entire extract takes up only 25 megabytes of disk space. VantagePoint is packaged as a set of class libraries, which are meant to be used with a Java development environment. Therefore, the installation program doesn't create the start-menu icons you'd typically find with other Windows-based installations.

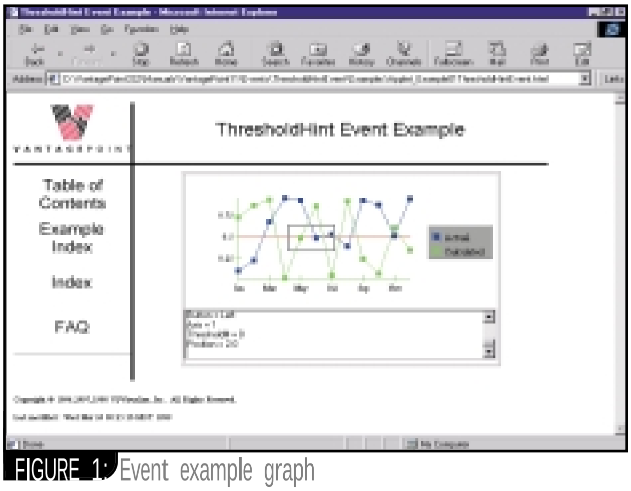


FIGURE 1: Event example graph

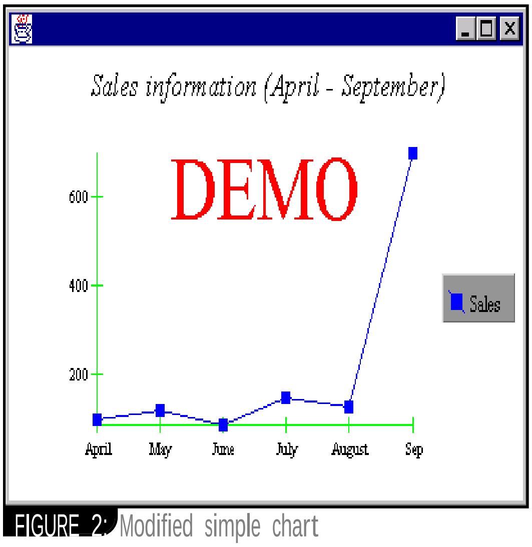


FIGURE 2: Modified simple chart

Working with VantagePoint

The easiest way to get started is to navigate to the root directory for VantagePoint and to open up the "index" HTML page, which will take you to a link for the online manuals. Within the manuals you'll find a wealth of example charts and graphs, such as the one shown in Figure 1.

All of the examples can be run directly from the example HTML pages. Visualize cleverly includes a listbox window in which you can see the events firing as you interact with the graphs. Since I'm not intimately familiar with the ins and outs of data visualization, I found these interactive examples to be an excellent way of learning how to work with the VantagePoint libraries. Visualize has a sparse tutorial, but I was able to use it to get my first program up and running fairly quickly. When you download VantagePoint you'll receive a license key via e-mail from Visualize. In order to run any of your own code you'll need to load the license as a parameter using the addLicense method of your charting object. I loaded the two-dimensional line chart sample program source code into the NetBeans Java IDE and edited the sample source code. I then added my license key to the chart object and modified the data values and parameters for this sample line chart. The resulting months-and-orders graph is shown in Figure 2. While it isn't terribly sophisticated, you have to appreciate the fact that I had the code modified and running in less than 10 minutes.

Summary

VantagePoint comes equipped with some very sophisticated routines for managing data and calculations and it will take you some time to become familiar with all of the capabilities of the product. This is not a product for a novice developer, but I'd recommend it for those developers that are looking at creating complex interactive visualization applications.

Splitting Tiers

Focusing on the front-end and middle tiers, and some alternative technologies

WRITTEN BY
AJIT SAGAR

The story about how the n-tier architectures evolved from the single-tier mainframe model has probably been told umpteen times by now (in fact, I retold it myself in last month's *e-Java* column). Nowadays the trend is to distribute functionality. Modularize everything. Components provide the means to successfully replicate your product in a gazillion scenarios. Client/server is old news. Think distributed architectures. Personalized Webtops. That's the name of the game today.

It's easy to get caught up in the hype and lose touch with reality. The Internet fosters a new type of dynamics that encourages and embraces change faster than the stability of technology. It's an ever-increasing challenge for developers and architects to make decisions regarding when to adopt a new technology and how long to retain the old one. The complexity of these decisions is compounded in that computing technologies usually span multiple tiers of an *n*-tier application. Development managers and project managers face their own problems when dealing with distributed architectures. One of them is how to structure the development team such that the skillsets of the team members map across the various technologies used to implement the business solution. Who is the architect and how does he or she keep a handle on both ends of the spectrum – from the back office to the front-end browser? What kind of middleware expertise is needed? What kind of GUI needs to be built and how much are the tools going to cost? How does the team responsible for the middleware interact with the front-end Web developers?

One way to alleviate the risk in making such decisions is to clearly define what the tiers of the architecture are going to be and how the functionality of your application maps to those tiers. Everybody talks about *n*-tier architectures and distributed architectures. But the bottom line is – How do you serve up data to your clients? This month in **e-Java** we'll start from where we left off last month. We'll revisit the four basic tiers of a distributed e-commerce application. Our focus will be more on the front end and middle tiers, and on some alternative Java and Web technologies available in the market to implement functional modules in business applications.

The Four Tiers

A typical distributed architecture that nurtures e-commerce applications consists of four-tiers:

- *Client user interface tier*
- *Presentation tier*
- *Business logic tier*
- *Data access tier*

Table 1 describes the purpose of each tier in an e-commerce application. In today's Web applications all of us strive to make the client as thin as possible. One of the main reasons for this is that end-user devices today range from desktop PCs to handheld PCs, cell phones and pagers. Thus, in the optimum distribution of tiers, the client UI tier will have minimum functionality; its only purpose will be to render the data on the screen using the display device's native graphic utilities. The thinnest "universal" client in today's distributed systems is the browser.

After All, It's All HTML

When it comes to the browser-based UI, all the data that the browser renders is interpreted from HTML. This means that the next tier down in the distributed system serves up HTML to the client. Traditional HTML serves static data only. To provide the high degree of interactivity typical of e-commerce transactions, HTML allows Web developers to embed tags that direct the browser to get fresh data from the server. Why do we have to go to the server to get data? Because the thin client doesn't do anything else but display data; it doesn't produce data. That's the job of the presentation tier.

Java applets provide a way to dynamically generate HTML content. However, applets are downloaded to the client and executed there, thus making the client thicker. Applets enable highly interactive behavior for the client and are good for calculations that can be conducted on the client side. But more complex computations that involve frequent data access (and require security) on the server side need to be executed in the next tier. This is the presentation tier and its responsibility in our context is serving up HTML to the browser-based client.

Splitting the Middle Tier

Thus dynamic content is generated by the tier next to the client UI. This tier, called the *middle tier* in traditional three-tier systems, could basically access the data from the back-office system, apply business logic to it and serve up HTML to the client. So why split this into a presentation and a business logic tier? Let me answer that question with another question. How do you get data from your data source that resides in the back office? The data could be obtained from sources of various types – database server, file servers, LDAP servers or some legacy system's proprietary interface. The result is that the means of getting the data will differ. Depending on the back-office interface, the middle tier's connection to the back office could be via DCOM, CORBA, RMI, raw HTTP or a variety of other protocols. The scope of a transaction could be

TIER	PURPOSE
Client	Renders data to user device using the device's graphic or other user interface utilities; interacts with end user
Presentation	Gets user input, parses it, submits it to business logic tier, gets results back from this tier, sends them to client
Business Logic	Gets data from data access tier through access mechanisms, applies business logic relevant to transaction, submits data to presentation layer; responsible for retaining state of transaction and workflow
Data Access	Gets data from back office, sends it to business logic layer in response to a query; also responsible for executing updates initiated by client

TABLE 1 Purpose of each tier in an e-commerce application

very different, depending on the business logic defined in the middle tier. If it's the same logical layer that deals with presentation logic as well as business logic, it becomes very complex. Also, since it's tightly coupled with the back office, it becomes less flexible and nonportable. This leads to scalability and performance issues.

The popular approach to solve these problems is to allow multiple application objects in a business logic layer. These application objects are responsible for getting the data from the server and making it available to the presentation layer. The presentation layer generates the dynamic HTML and makes it available to the client browser.

Generating Dynamic Content

The presentation layer has several alternatives for generating the dynamic content and serving it back to the client. The most popular approaches are via server-side scripting, Java servlets and Java Server Pages (JSPs), and Active Server Pages (ASPs). ASPs, Microsoft's technology for accessing COM objects, are beyond the scope of this article. JSPs are Java's counterpart to ASPs (hence the name *JSP*) and are Sun's technology for accessing Java objects on the server. All services on the servlets are Java's gateway to all services on the server side. Servlets are always associated with a Web server. They're created as server threads as a result of a client's HTTP request and they serve up HTML content as a response. A JSP allows developers to embed Java code in HTML pages. These hybrid pages have a .jsp extension and can contain a combination of HTML, Java code and JavaBean components. When the client makes an HTTP request, the servlet engine on the server compiles the JSPs into corresponding servlets.

Another way to add dynamism to HTML is via server-side scripting using JavaScript or JScript. Scripting code is embedded in the page. The difference is that, unlike JSPs, scripting code isn't compiled, it's interpreted. Although this is slower, it's faster to develop, prototype and test. Another popular middleware technique for generating dynamic HTML is to use extensions to HTML tags, which allow the page to access dynamic content. This technique is offered by Web application programming languages like Allaire's CFML.

These presentation technologies are still needed to talk to the business logic tier in order to actually obtain the data that needs to be presented. That is where server-side technologies like EJBs and COM come into the picture. The business logic and data source accessibility is abstracted by these technologies.

E-Commerce Functional modules

The technologies selected for your application should be based on the behavior that's expected from the functional module. For example, you can create a shopping cart component using CGI, JavaScript, servlets, JSPs, CFML or other types of dynamic-content generation technologies. However, when settling on a technology, you should ask yourself what the interface to the business logic tier is. Are your business objects COM objects, EJB objects or just database adapters? Can you access the COM objects via CORBA? Is there even a need to talk to the business logic tier for all transactions? For example, if the next step in your workflow is a credit card payment, the information doesn't need to go beyond the presentation layer, provided that the presentation layer has hooks into a credit card payment system like CyberCash.

One of the unique features offered by technologies like CFML is that it allows you to embed SQL inside your HTML. This means you can access the database directly from your HTML page as opposed to going to a business logic layer. This allows you to segregate your database into a front-end database, which could contain, for example, user profiles, account information and maybe local catalog information. On the other hand, if you need to access services offered by the back office such as information pulled out in real time from a reservation engine, you probably need to call Java or COM objects on the server side. In addition, if your catalog information is dependent on business rules that tie into the back office, then you'd probably like to go through a business logic tier. In that case JSPs may be a better alternative to build your shopping cart.

Trading Places

The notion of splitting the middle tier is important for building robust and flexible business applications. You may also want to divide the responsibility of load balancing and fault tolerance between the two middle-tier layers (presentation and business logic). For example, both ColdFusion (presentation) and WebLogic (business logic) offer clustering capabilities. Some transactions, such as credit card payments or catalog queries, may not need to go back to the business logic tier. Decoupling these two layers gives a large boost to scalability.

In the end, what the end user sees is your workflow that abstracts him or her from all these complexities. However, as someone who provides e-business solutions, I believe it's our job to create these abstractions to offer a more satisfying experience for the users of our systems.

E-Book

Starting this month, I'd like to add a monthly book review to the **e-Java** column, an idea I am shamelessly stealing from Alan Williamson of **Straight Talking** fame. Having received mail from several readers who have asked about good sources of information on several topics, I figured this is a good way to share my opinion on recently released books with all you fine folks.

In the mad rush for deliverables and the race to overcome the time-to-market dilemma, I find myself more often than not desperately trying to catch up on technology. In Java, that roughly translates to new APIs. Since I returned to Dallas pumped up from JavaOne in June, I've been trying to find time to catch up on J2EE. O'Reilly & Associates' Nutshell Handbook series seems to be created for such situations. *Java Enterprise in a Nutshell: A Desktop Quick Reference* is a great book if you want to get a quick ramp-up on the J2EE APIs. It starts off where *Java in a Nutshell* ended. This is an excellent reference, but it's not for everyone. If you're looking for a comprehensive Java reference that covers Java in all its glory, this isn't the book for you – try *Core Java* published by Prentice Hall. If you're looking for a book on Java 1.1 APIs, don't start with this one – start with its predecessor.

If you know core Java, however, and are interested in a quick introduction to the APIs provided by the Java platform for developing enterprise applications, make sure you get a copy of *Java Enterprise in a Nutshell*. In typical "nutshell" style, Part I starts with a quick introduction to the Java enterprise APIs, and also gives a concise explanation of what enterprise computing means and how it relates to Java. The diagram on page 12 and the corresponding discussion in chapter 1 are instrumental in bringing readers up to speed with the current role of Java APIs in an e-commerce application. Part II covers the J2EE APIs – JDBC, RMI, Java IDL, Java servlets, JNDI and EJBs. There's just enough code to get your feet wet. If you want to look up an API, read the corresponding chapter, play around with the examples and then go write your application. For more detail and complex examples you'll have to get a book that focuses on that particular API. Finally, Part III of the book is a comprehensive quick reference for the enterprise APIs.

All in all, this is a good book to add to your library. And it's light enough to haul along if you have to travel.

ajit@sys-con.com

Author Bio
Ajit Sagar is a member of the technical staff at i2 Technologies in Dallas, Texas, focusing on Web-based e-commerce applications and architectures. A Sun-certified Java programmer with nine years of programming experience, including three in Java, Ajit holds an MS in computer science and a BS in electrical engineering.

The Oracle at Boulder

The year of the application server boom is just about over. What happens next?

WRITTEN BY
JASON WESTRA

I was asked to stick my neck out and write on the future of Enterprise JavaBeans in year 2000. Just so you know, I was one credit away from a minor in the classics (you know, Greek mythology, ancient Rome and Egypt). However, since I *didn't* major in this field, nor even minor in it, don't hold me to anything I say for I'm no Oracle at Delphi. I'm just a soothsayer from Boulder, Colorado, and even then I'm only one of many (though most in Boulder foretell fortunes based on constellations, tarot cards or your sign!).

This month I'll give you my gut feelings on what's in store for EJB next year. I hope to enlighten you, but suspect that anyone following the EJB rage over the past year or so has come up with some of the same conclusions.

Y1999 — Where Are We Coming From?

To know where we're going, we need to take a look at where we're coming from. So let's gaze into the past year or so to understand how EJB got to where it is today.

I consider (and I'm not alone) 1999 the year of the application server boom. There were three key catalysts driving this movement: Microsoft's component model COM/DCOM (including COM+) matured, XML's importance for EAI increased and, most of all, Enterprise JavaBeans became the server-side component model of choice for Java developers worldwide. In 1999 it seemed as though any company that had ever done server-side Java in the past became an application server vendor offering an EJB solution.

To my way of thinking, no other technology has influenced the application server boom as much as EJB. The enormous interest derives from the fact that its component model shelters developers from the need to develop low-level services such as distributed object communication and location, transaction support, resource and thread allocation, and even persistence of enterprise beans. In addition, EJB developers work with a fun, portable platform and focus on the business logic of their applications, thereby increasing time-to-market of their products.

In 1999 we saw numerous EJB products released, including EJB servers of varying degrees, third-party EJB containers, and code generators and deployers bundled with various component modeling tools and Java IDEs. With this exciting support for EJB in 1999, it looks as though EJB is on the verge of greatness in year 2000.

Y2000 — Where Are We Going?

EJB is maturing at a rate faster than anyone could have guessed…or hoped for. This pace of growth influences my belief that in year 2000 the following will occur:

- The flooding of the EJB server market will subside, leaving a few dominant players controlling a large portion of the market.
- The market for reusable third-party components will be realized (but not fully).
- The server-side component model wars that began in 1998 and increased in 1999 will escalate to a point at which there may be a declared winner.
- An increase in demand for Enterprise JavaBeans technology will cause a shortage of EJB talent, fostering a demand for training and skills in distributed computing.

Market Share of EJB Servers

When ***JDJ*** editor-in-chief Sean Rhody last counted, 28 companies with application servers were offering EJB support (thanks for the legwork, Sean!). Depending on when Sean counted his beans, this number is perhaps one less: Forté Software, Inc., recently merged with Sun Microsystems. The merger is an example of times to come as more and more small fish in the EJB server sea consolidate with larger, established companies. While the number of EJB servers may continue to increase in the year 2000, I believe that 80% of the market for EJB servers will be controlled by BEA Systems, Inc., IBM, Oracle and Sun Microsystems (the Big 4).

I don't think that all 28 vendors offering EJB support are really battling for a share of the EJB server market. Rather, they include EJB support as a necessary feature to sell their application server, which really may have been built to solve other problems, such as data integration EAI and dynamic Web content management. For example, Novera Software, which used to call its flagship product the Novera jBusiness Application Server, changed its name to the Novera Integration Server. While datasheets mention its EJB support, the NIS is really marketed for its ability to tie disparate data sources together in a unified business component view – not its EJB prowess.

In 2000 those companies that are really battling for a portion of EJB's lucrative server business will gain market share based on three criteria: current market position, technology/implementation beyond J2EE, and runtime monitoring and deployment facilities. Each is discussed in depth below.

1. CURRENT MARKET POSITION

The Big 4 win here. As previously stated, the big fish of the EJB server market that I'm referring to are BEA, IBM, Oracle and Sun. These firms hold a major portion of the market now, and will

increase their market share through reputation and sales of existing products, or through an acquisition/merger situation with companies offering similar products (e.g., BEA Systems' purchase of WebLogic and Forté's merger with Sun). As a longtime Forté user, I'm glad to see such exposure for its technology and I believe it will become a key part of Sun's EJB solution offering.

To demonstrate how powerful current market position is in winning the battle for EJB server market share, let's take a look at Oracle Corporation and why I believe it will prosper as a member of the Big 4.

I assume Oracle's server-side Java strategy is to put an EJB server in every Oracle database on the planet. Considering it holds the largest market share of enterprise databases in the world, Oracle could soon dominate the market in EJB servers as well. Not convinced? This strategy is nothing new and may be termed *gorilla marketing*, a tactic that has made more than one technology company an industry maker. Take Microsoft, for example. It gained major headway in the browser wars by including its Internet Explorer browser free on every PC.

2. *TECHNOLOGY/IMPLEMENTATION BEYOND J2EE*

EJB servers and tools have matured at a rapid rate. All provide the basics such as database connectivity, naming and directory services, life-cycle management and transaction management. These services are being bundled into the Java 2 Enterprise Edition, and, as you may know, EJB is the foundation upon which J2EE is built.

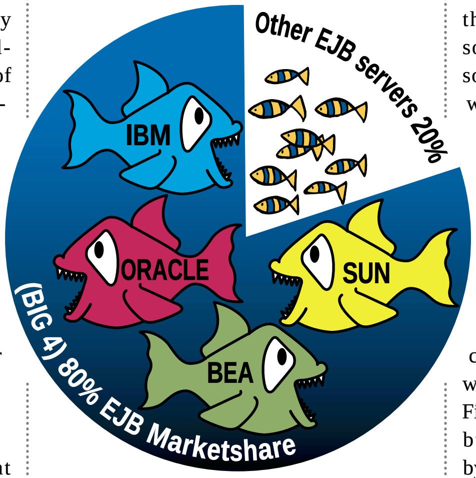
J2EE compliance is a must for any application server serious about competing as an enterprise Java server. Thus, winning market share will be governed by a vendor's offerings in the areas of performance and scalability. The pressure is on for EJB servers to provide load balancing and fault tolerance through clustering or replication of components.

Even then, I don't believe a highly scalable clustering architecture alone will be enough for an application server to dominate the market. The application server must also provide an environment that is easy to develop enterprise beans against and one that makes deploying these components easy. This leads to the last of my three criteria for a successful EJB server in the coming year.

3. *RUNTIME MONITORING AND DEPLOYMENT FACILITIES*

The ability to manage increasingly complex deployment configurations will be a deciding factor in sales of application servers in the year to come. Successful EJB vendors will have to provide rich runtime monitoring tools that are extensible, allowing you to customize them to your application's needs.

For instance, system administrators need the ability to create custom system agents that track user-defined statistics in a graphical console environment. Runtime monitoring capabilities will need to allow system administrators to chart activity, start and stop distributed components, roll new versions of enterprise beans into active servers without downtime, and even add or remove servers from a cluster as performance dictates.



Agents should also be autonomous, acting on system variables automatically for the system administrator. Last, seamlessly tying an EJB server's monitoring capabilities into other enterprise management systems such as Tivoli will be a value-add feature that will set an EJB server apart from the pack in Y2K.

Besides rich runtime monitoring capabilities, I believe EJB servers that demand developers to hand-build UNIX scripts or .bat files for deployment will need to advance their application management and deployment facilities to sucessfully compete in the EJB server race next year. Products with intuitive, graphical deployment environments will be the rage in the year to come as more and more nontechnical Java developers become Enterprise JavaBeans developers and deployers. A graphical deployment environment for components is a strong suit of Forté's SynerJ Application Server and Deployer product lines.

Like the Java platform itself, Enterprise JavaBeans is an expansive playing field with marketable opportunities in servers, containers, code generators, deployers, third-party components and training. Let's take a gander at how a few of these markets will evolve in the year 2000.

Business Components on the Rise in Y2K

My background is a mixture of consulting and product development. For three and a half years before I joined Verge, I worked for the CSC Lynx Group, a product development organization within CSC Consulting. The business model for the CSC Lynx Group was to produce component-based frameworks for distributed, high-volume systems that the consulting arm of the organization would use to rapidly implement business solutions.

The CSC Lynx Group was always ahead of its time, implementing into its frameworks core services such as load-balancing algorithms and custom fault tolerance that weren't provided by the distributed software environments or middleware solutions available then. These services were needed in our frameworks to provide reliability and scalability, and to support heavy transaction volumes. While the frameworks were implemented with various technologies, when it came time to do the same with Java in 1998, EJB seemed to offer most of the value-add that our frameworks had provided in the past. The core services we were used to building were already handled by the EJB server. Finally, the group was able to "get down to business" and provide real value to clients by developing pure business solutions!

I think next year we'll hear much less talk about "distributed frameworks" and more about developing interoperable business components. This includes third-party business components built for a variety of e-business markets such as online banking and securities trading. Companies are already providing business components to rapidly develop EJB solutions. A few to put on your radar screen are The Theory Center (www.theorycenter.com) and, of course, IBM's San Francisco. More names of companies and their components will be added to this list in Y2K.

The demand for third-party components will increase in 2000, and more companies will enter this market as component providers. However, I'm not convinced that these components will truly be considered off-the-shelf software. Business components, no matter what their implementation, are complex to configure and customize without proper guidance. Consulting and professional services groups from the component suppliers will need to provide their expertise to properly configure and deploy their EJBs at your organization. As server-side business components become more commonplace, perhaps in a few more years, you'll be able to buy a business component at the local software store and install it yourself – but not next year.

Off-the-shelf components aren't a new concept. In fact, a war around client-side components began when the JavaBeans specification was released. JavaBeans became a true contender for Microsoft's dominance of the market with its OCX and ActiveX technologies. Now the war has moved to a new front. It's moved to the server where I predict the war for server-side components will peak in year 2000.

Server-Side Component Model War Escalates

Y2K will be the year that makes or breaks Enterprise JavaBeans as the server-side component model of choice for corporate developers. I believe there are two competitors in this war: DCOM and EJB. I don't see CORBA as a solution for new component development, but rather a way to integrate legacy code into new solutions built around EJB.

I admit I have some reservations about EJB's ability to win the war against Microsoft's COM/DCOM solution. COM offers much of the same functionality as EJB around component-based development, and COM+ includes features such as component replication – and load balancing to provide fault tolerance – allowing Microsoft solutions to scale to an enterprise level. The caveat remains concerning the ability of COM to operate on multiple platforms, which has already been done on Solaris and will soon be done on other UNIX platforms. Regardless of COM's strides in platform portability, it's unrealistic to deploy a COM-based application into production on anything but Windows 95 or NT.

In the end, the server-side component model war won't be measured by the component model with the superior technology that each camp will always claim ownership of. Many top-notch technologies have come and gone with less success than they deserved, considering the competition – the Macintosh, for instance. Tangible evidence, such as the number of trained professionals and the number of successful business applications in production will be the true indication of a winner in the server component war. That said, I stress the importance of EJB training in year 2000.

Train the Troops

I foresee many doomed EJB projects in the coming year, and not because the technology and tools weren't powerful enough or mature enough. Actually, I suspect that most failed EJB projects will result from a lack of EJB talent (and distributed computing experience in general). Next year, IT managers and developers alike will embrace Enterprise JavaBeans by the hoards for its ease of use and component-based development. A rude awakening awaits anyone who believes distributed components are easy! Often, the blame for missed deadlines or failed projects is placed on the technology rather than on the lack of training or experience in it. Also, remember that training can't sufficiently provide what years of experience building distributed systems can; thus training from vendors on their EJB products may provide a false sense of security. Before starting your first EJB project next year, train your troops and hire experienced developers!

Conclusion

I look forward to seeing the impact EJB will have on organizations and software development as a whole in the year to come. EJB will continue to revolutionize enterprise development, either by itself or alongside other technologies such as XML and CORBA. I leave with one last expectation about EJB in year 2000…. While the Oracle at Boulder may be wrong on all counts above, I *am* sure of one thing: Microsoft will not embrace EJB as a server-side component model. The war will wage on. Mark my words.

jwestra@uswestmail.net

Author Bio
Jason Westra is a managing partner with Verge Technologies Group, Inc., a Java consulting firm specializing in JavaBeans solutions.

JDJ FEATURE

The Java 2 Enterprise Edition

J2EE promises the very characteristics needed for inexpensive, rapid development and deployment of e-business applications

WRITTEN BY GREG FLURRY

Sun, IBM, Novell, Oracle and nearly 50 other companies have proposed the Java 2 Platform, Enterprise Edition (J2EE) as a solution for the development and deployment of e-business applications. What is J2EE? What does it offer to developers and users of e-business applications? This article answers these questions and provides a sample application built on J2EE principles.

The Promise of J2EE

As more business is conducted over the network, enterprises find they can achieve more with less; they can interact with their customers and business partners more quickly and cheaply using networked business-to-business and business-to-consumer applications. These enterprises are conducting e-business. J2EE promises to make e-business even more compelling by defining the means to quickly and inexpensively develop, modify and deploy portable, easy-to-use, reliable, interoperable, scalable and secure e-business applications. To fulfill the promise, J2EE defines the following elements:

- A platform, including a runtime environment based on industry-standard APIs and protocols, and a well-defined scheme for packaging and deploying multitier e-business applications.
- An application programming model for developing applications that leverage the platform.
- A compatibility test suite verifying that a J2EE platform implementation complies with the J2EE platform definition; successful completion of the test suite serves to assure application developers that their J2EE-compliant applications will deploy and run on that J2EE platform implementation.
- A reference implementation that intends to provide an operational definition of the J2EE platform. It provides concrete answers where the platform and application programming model definitions are open to interpretation, and also helps popularize J2EE, serving as a means of demonstrating the capabilities as well as a base for prototyping J2EE applications.

While I was writing this article, a public draft of the specification for the J2EE platform was available, and a beta draft of the J2EE application programming model specification and a beta version of the reference implementation had just become available. A beta form of the Compatibility Test Suite is expected before the end of 1999. See htttp://java.sun.com/j2ee for details.

I'll focus on the J2EE platform and its benefits, and offer a brief look into the application programming model. I'll also describe an application that demonstrates the principles of the J2EE application programming model.

The J2EE Target Environment — the Middle Tier

Enterprise-class e-business applications typically exhibit multiple logical tiers, as shown in Figure 1. Depending on the application, the client tier may provide only a user interface or may also run some business logic. The client communicates with the middle tier via the Internet, an extranet or an intranet. The middle tier generally performs some or all of the user interactions and runs some or all of the business logic. The Enterprise Information Services (EIS) tier encompasses legacy or new, business-critical data and applications that typically should be accessed only by the middle-tier business logic – for example, databases, ERP systems and transaction systems.

J2EE focuses on the middle tier, which forms the foundation of an environment that must be secure, highly available, scalable and manageable. Figure 2 shows how J2EE addresses the middle tier. A Web server, enhanced to support Java servlets and JavaServer Pages (JSP), handles the user interaction. An application server running Enterprise JavaBeans performs the business logic and accesses the EIS on behalf of the user. (See http://java.sun.com/j2ee for more information about servlets, JSPs, EJBs and other aspects of enterprise Java and J2EE.)

Figure 2 shows that J2EE allows a wide diversity of clients and protocols between the multiple logical tiers of the e-business application. J2EE makes recommendations, however, on the preferred application model for clients based on business needs and customer requirements. From the Internet or intranets, J2EE anticipates HTML and HTTP clients. The former, such as Web browsers, communicate with the Web server via HTTP or HTTPS and send and receive only HTML for presentation to a user. HTTP clients also communicate with the Web server via HTTP or HTTPS, but can exchange richer information, such as XML documents. These clients would typically be built using Java-based applets or applications. In the intranet, although HTML and HTTP clients are preferred, J2EE also anticipates IIOP clients that communicate through the application server to interact with EJBs directly. Today such clients must be Java-based applets or applications; future clients may include Microsoft COM objects.

The J2EE Platform

The J2EE-platform specification defines a runtime architecture for J2EE-compliant applications and defines mechanisms for application assembly and deployment. The J2EE runtime architecture, shown in Figure 3, makes a clear distinction between the J2EE platform – that is, the runtime infrastructure for an application – and the application itself. The platform consists of a set of containers: an applet container for client-side applets, a Web container for the servlets and JSPs, and an EJB container for the EJBs. The applets, servlets, JSPs and EJBs that constitute an application are called *components*.

The distinction between containers and components is very important. A container provides a standard runtime environment for a component, that is, a set of services the component can use to accomplish a specific task. The component services define a concrete set of application programming interfaces that enable components of the proper type to access underlying J2EE services. A container can therefore transparently provide enterprise-critical services such as transaction management and security while simultaneously providing enterprise-critical attributes such as scalability, reliability and support for multiple users. These container properties make the task of writing components much easier and guarantee that components written by a component provider can be portable between J2EE platforms from different J2EE platform providers.

The set of J2EE services provided to a component by a J2EE container depends on the component type. The most basic applet container provides only the Java 2 Platform, Standard Edition (J2SE), which includes JavaIDL and the JDBC base (see http://java.sun.com/products/jdk/1.2). A more sophisticated applet container – for example, one that allows applets to access EJBs directly – provides additional enterprise APIs, such as JNDI and RMI/IIOP, as shown in Figure 3.

The Web container always provides the J2EE server core, which includes J2SE and the following enterprise APIs: a JDBC extension, RMI/IIOP, EJB client, JMS, JNDI, JTA, JavaMail and JAF. In addition, the Web container always provides the enterprise APIs for servlets and JSPs. The presence of JDBC requires that the Web container also provide a database service.

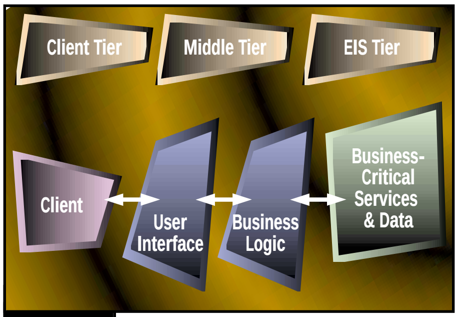


FIGURE 1 Generic e-business application architecture

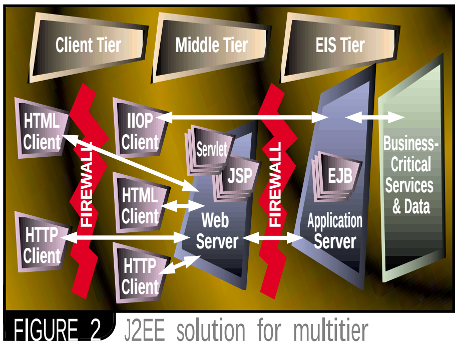


FIGURE 2 J2EE solution for multitier e-business applications

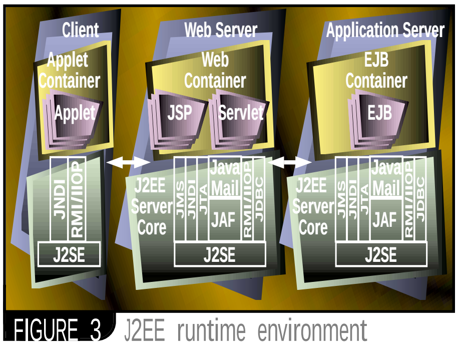


FIGURE 3 J2EE runtime environment

The EJB container always provides the J2EE server core and the enterprise APIs for EJB servers. It also provides a database and a transaction processing service to support the persistence and transactional capabilities of EJBs.

Any J2EE platform must supply all the containers, services and APIs required by the J2EE platform specification, eliminating the need for application writers to deal with tricky integration issues. Each version of the J2EE platform specification will require specific versions of the APIs. Future versions of the J2EE platform may require support for additional APIs.

One of the clear goals of J2EE is interoperability with a number of different clients. To achieve this, the J2EE platform services and APIs support a set of industry-standard communication protocols and data formats. J2EE requires support for the following protocols: TCP/IP and UDP/IP, HTTP, SSL (HTTPS), IIOP and JRMP. J2EE requires support for the following data formats: HTML, GIF, JPEG, and Java JAR and class files. Each version of the J2EE platform specification will require specific versions of the protocols and data formats. Future versions of the J2EE platform will require support for additional standards, such as XML.

Figure 4 depicts the J2EE platform specification for application assembly and deployment. Application assembly begins with the creation of components, such as servlets, JSPs and EJBs, using the APIs, protocols and data formats mentioned above. Multiple components for the same container type are packaged into a module that requires a module deployment descriptor containing information on how to deploy the components and how these components help compose an application. The module is the smallest unit for deployment on J2EE platforms. A J2EE application includes one or more modules and an application deployment descriptor that describes the top-level view of an application – all the modules and information including application-level security roles. The final step in packaging the J2EE application is the creation of an enterprise Java archive file.

Deployment of a J2EE application on a J2EE platform leverages the platform's deployment tool to dearchive the application package, examine all the modules in the application and install the various components from the modules into the appropriate containers. Finally, the administrator must configure the component containers with deployment-specific values for all the properties in the various deployment descriptors, including security roles, user lists, transactional attributes and database targets.

A significant differentiator of the platform is the capability to assign different roles to different individuals or groups. J2EE platform providers offer the runtime environment including containers, APIs, enterprise services, and deployment and management tools. Component developers can create J2EE components, packaged as modules, that are independent of any particular platform implementation. Application packagers can create and sell applications using components provided by multiple component developers. A deployer installs and configures a J2EE application so it runs correctly and securely in a particular J2EE platform. A system administrator configures and manages the J2EE platform to make sure it provides an efficient and scalable runtime environment for applications. Even though the lines between the different roles may be blurred in some cases, this division of labor made possible with the J2EE platform increases the manageability of the platform itself, as well as the portability and manageability of applications deployed on the platform. Further, it creates multiple opportunities for providers of e-business applications and application components and tools, and helps lower the cost of creating, deploying and using e-business applications.

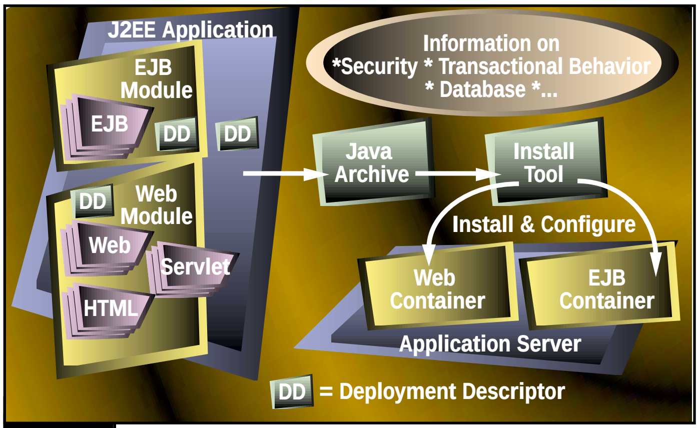


FIGURE 4 J2EE application packaging and deployment

J2EE Application Programming Model

The J2EE application programming model (APM) recommends how the J2EE platform specification should be applied to various application domains to derive the maximum benefit from the J2EE platform. The APM aims to make it easier to design manageable, deployable and maintainable e-business applications and get them to market quickly.

Figure 2 offers a high-level view of the J2EE APM. For the client tier of a J2EE application, the APM focuses primarily on a client that accesses a Web server using HTML (and later XML) over HTTP or HTTPS; however, the APM recognizes the likelihood that intranet clients will use IIOP to access the application server directly. In the middle-tier Web server the application uses JSPs and servlets for various forms of user interaction and control. In the middle-tier application server the application uses session EJBs to represent execution objects and uses entity EJBs to represent data or application objects.

Figure 3 offers a low-level view of the J2EE APM. At a low level the client-tier components use J2SE and perhaps some enterprise APIs to access the middle tier. The servlets, JSPs and EJBs in the middle tier use J2EE server core APIs and other enterprise APIs to access various enterprise services.

Since the APM is already defined at a high and low level by the J2EE platform specification, you should consider the J2EE APM specification to be a collection of architectural guidelines and design patterns for optimally developing and deploying e-business applications targeted at J2EE platforms. The APM specification addresses a number of issues, including:

- Using servlets for processing client requests (controller) and JSPs for client responses (views)
- Accessing services directly or through "access objects" and "data access objects" abstractions
- Accessing EIS resources via EJBs or directly from JSPs
- Using session or entity EJBs
- Using distributed transactions versus local transactions
- Accessing external systems with "connectors"
- Using a model-view-controller design pattern
- Using JSPs or servlets in the Web container
- Accessing services directly or through "access objects" and "data access objects" abstractions
- Accessing EIS resources via EJBs or directly from JSPs
- Using session or entity EJBs
- Using distributed transactions versus local transactions
- Accessing external systems with "connectors"
- Applying security measures

A Sample Application — the Freeside Bank

IBM, as a supporter and contributor to J2EE, created a sample application that demonstrates important aspects of the J2EE platform. The Freeside Bank application, shown at JavaOne this year, is a prototypical banking scenario that includes activities such as customer login, account balance display and funds transfer. Freeside assumes a browser-based HTML client, uses JSPs to perform user interaction, and uses entity EJBs with container-managed persistence to represent customers and accounts and to perform transactions. Freeside is a good approximation of a J2EE application and is useful to illustrate J2EE concepts, even though the application was implemented before the J2SE and J2EE specifications solidified.

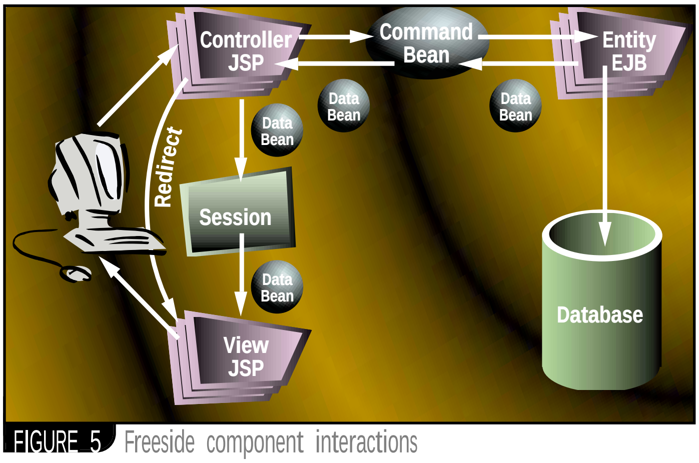
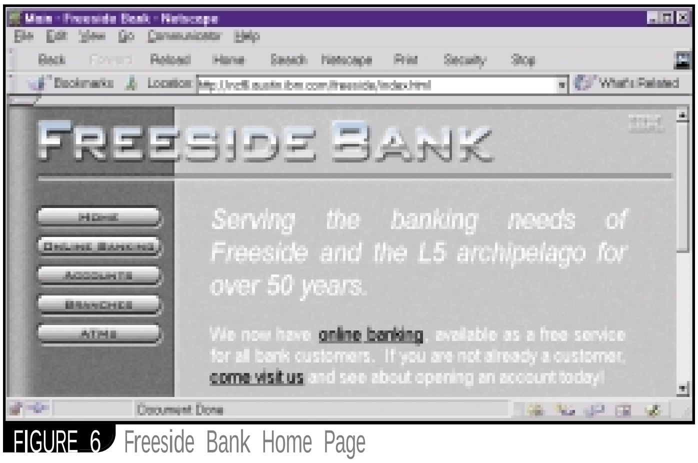


FIGURE 5 Freeside component interactions


Main - Freeside Bank - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks Location: http://ncf6.austin.ibm.com/freeside/index.html
What's Related
HOME
ONLINE BANKING
ACCOUNTS
BRANCHES
ATMS
Document: Done

FREESIDE BANK

Serving the banking needs of Freeside and the L5 archipelago for over 50 years.

We now have online banking, available as a free service for all bank customers. If you are not already a customer, come visit us and see about opening an account today!

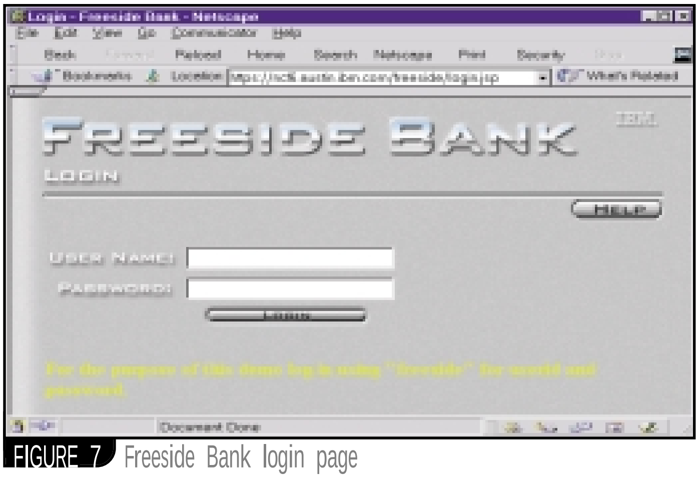
FIGURE 6 Freeside Bank Home Page

In addition to demonstrating J2EE platform components, Freeside introduces the useful "command" design pattern, which is somewhat analogous to the "access object" design pattern in the APM specification. Briefly, the command pattern calls for the user of a command bean to instantiate the command bean, set the input parameters required to run the command's business logic, and then request that a command manager execute the command's business logic. The command manager incorporates knowledge about where to run the command's business logic, either locally to the client of the command or remotely. In either case, the client then locally accesses the results of the business logic execution. The command pattern presents a single model for local and remote command execution; for remote execution, commands result in fewer distributed method calls, greatly reducing the overhead in setting the input parameters, executing the business logic and retrieving results.

You can download the source and executable code for Freeside from www.developer.ibm.com/tech/integration/jsp.html. If you wish to run Freeside, you'll need Windows NT 4.0; JDK 1.1.6 or later; IBM's WebSphere Application Server, Advanced Edition 2.0 that includes IBM's HTTP Server v1.33 and IBM's DB2 v5.2; and a Web browser (4.0 level or higher, JavaScript-capable). You can find a free trial version of WebSphere at www.software.ibm.com/webservers/download.html.

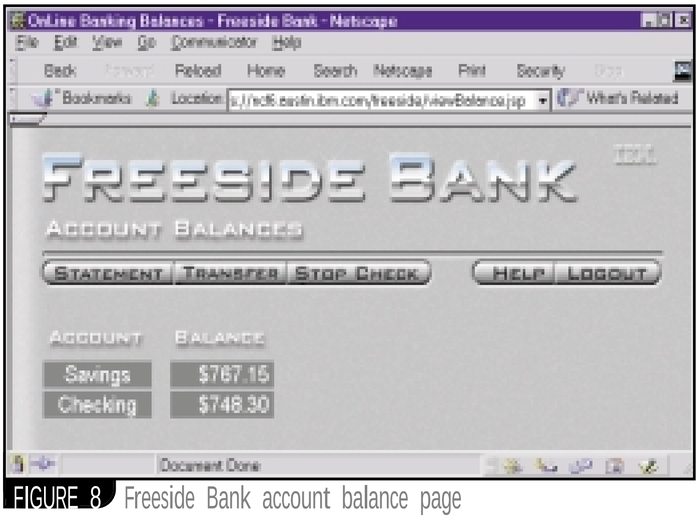
Figure 5 shows the typical actions for a client request in Freeside and gives you a feel for how JSPs, commands and EJBs work together in a J2EE environment. The client makes a request on a controller JSP; the controller JSP uses a command bean to access an entity EJB with container-managed persistence. The entity EJB represents information in a database. The EJB encapsulates its information content in a data bean that the command bean retrieves using a method on the EJB. The controller JSP extracts the data bean from the command and stores it in the Session object for the interaction. The controller JSP then redirects the client request to the display JSP, which then extracts the data bean from the Session object and performs the necessary activities to complete the request and display the results.

To make the description more concrete, I'll describe the Freeside customer-login process, starting from the Freeside "home page" (see Figure 6):


Login - Freeside Bank - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks Location: https://ncl6.austin.ibm.com/freeside/login.jsp
What's Related
FREESIDE BANK
IBM
LOGIN
HELP
USER NAME:
PASSWORD:
LOGIN
Document: Done

For the purpose of this demo log in using "freeside" for userid and password.

FIGURE 7 Freeside Bank login page


OnLine Banking Balances - Freeside Bank - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks Location: s://ncl6.austin.ibm.com/freeside/viewBalance.jsp
What's Related
FREESIDE BANK
IBM
ACCOUNT BALANCES
STATEMENT
TRANSFER
STOP CHECK
HELP
LOGOUT
Document: Done

ACCOUNT	BALANCE
Savings	$767.15
Checking	$748.30

FIGURE 8 Freeside Bank account balance page

- The customer presses the "Online Banking" button that invokes the login JSP (login.jsp).
- The login JSP displays a form (see Figure 7) that collects the customer name and password; pressing the "Login" button invokes another JSP (doLogin.jsp). This separates user interaction from control.
- Listing 1 shows the key logic for the doLogin.jsp; it creates a Login command, sets the customer name and password parameters for the command, and requests that the command manager execute the command to authenticate the customer.
- Listing 2 shows the Login command "business logic"; it first gets an EJB-Home object for the customer entity EJB, then attempts to find the customer in the bank's database using the customer name input parameter (customerLogin) as the key for the EJBHome finder method; assuming the customer exists, the EJBHome returns a customer EJB instance; the command then validates the password by comparing the password parameter (customerPW) to the password in the database, obtained by calling the getPassword() method on the Customer EJB; if the customer is validated, the command sets the output parameter (customerDataBean) to the bean returned by the Customer EJB getDataBean() method.
- Listing 3 shows the ejbCreate() and getDataBean() methods for the customer EJB; the EJBHome object for the customer EJB creates an instance and calls the ejbCreate() method when its search for the customer succeeds; ejbCreate() parameters are the information derived from the database by the EJBHome finder method; the getDataBean() method simply creates a data bean used to return customer information in one interaction rather than multiple interactions.
- Returning to doLogin.jsp in Listing 1, if the Login command succeeds, the JSP places the customer information returned by the Login command (and the Customer EJB) in the form of a data bean into the session object and redirects the browser to the viewBalance.jsp to display the account balances (see Figure 8); if the command fails, redirection is used to redisplay login.jsp.
- The viewBalance.jsp (see Listing 4) first gets the customer information in the form of a data bean from the session object, then retrieves account information by creating a ViewAccounts command, setting the parameters and asking the command manager to execute the command.

Listing 1: Key logic for login controller JSP (doLogin.jsp)

```
<%
String id = request.getParameter("username");
String password = request.getParameter("password");
Login loginCommand = new Login();
loginCommand.setCustomerLogin(id);
loginCommand.setCustomerPassword(password);
String urlString = JSPUtil.getBaseURL(request);

try {
  loginCommand =
    (Login) CommandManager.execute(loginCommand);
  HttpSession session = request.getSession(true);
  session.putValue("customer",
    loginCommand.getCustomer());
  session.putValue("transactionList",
    new Vector());
  response.sendRedirect(urlString +
    "viewBalance.jsp");
} catch (CommandException commandException) {
    response.sendRedirect(urlString +
    "login.jsp?customerID=" + id);
}
%>
```

Listing 2: Business logic of Login command (Login.java)

```
try {
  CustomerHome customerHome = (CustomerHome)
    EntityUtility.lookupHome(CustomerHome.class);

  Customer customer = null;
  try {
    customer =
      customerHome.findByLogin(customerLogin);
  } catch(ObjectNotFoundException ex) {
      throw new InvalidLoginException();
  }
  if(!customerPW.equals(customer.getPassword())){
    throw new InvalidPasswordException();
  }
  customerDataBean = customer.getDataBean();

} catch(Exception ex) {
   ex.printStackTrace();
   throw ex;
}
```

Listing 3: Excerpts from the Customer EJB (CustomerBean.java)

```
public void ejbCreate(String login,
                      String password,
                      String name)
  throws RemoteException, CreateException

 {
   this.login = login;
   this.password = password;
   this.name = name;
 }

public CustomerDataBean getDataBean()
       throws RemoteException
{
  CustomerDataBean dataBean =
    new CustomerDataBean(login, password, name);
  return dataBean;
}
```

Listing 4: Key logic for viewBalance.jsp

```
<%
  CustomerDataBean customerInfo =
    (CustomerDataBean)
```

- Listing 5 shows the "business logic" for the ViewAccounts command. The command finds an EJBHome for the account entity EJB and then finds all the accounts for a customer using the customerID parameter as a key; the finder returns an enumeration of account EJBs; the command puts the data beans from all account EJBs in its output parameter (accounts).
- Listing 6 shows the ejbCreate() and getDataBean() methods for the account EJB; the EJBHome object for the account EJB creates an instance and calls the ejbCreate() method when its search for an account succeeds. The ejbCreate() method sets the information derived from the database by the finder method; the getDataBean() method simply creates a data bean used to return account information in one interaction rather than multiple interactions.
- Returning to viewBalance.jsp in Listing 4, if the ViewAccounts command succeeds the JSP proceeds to display the account information for all the accounts owned by the customer.

Once Freeside has displayed the account balances, the customer can request additional actions, such as transferring funds between the accounts. These additional actions use JSPs, commands and EJBs in a manner similar to that described above. The transfer action is particularly interesting in that it uses several levels of JSPs calling JSPs, commands calling other commands, and involves several entity EJB classes. A detailed description is beyond the scope of this article; you can examine the Freeside source if you're interested in learning more.

Conclusion

The J2EE platform defines a complete, machine-independent, standard environment that supports applications requiring security, reliability, scalability, transactions and other enterprise-class attributes. The J2EE APM builds on the productivity inherent in Java technology by offering guidelines for quickly creating J2EE-compliant applications while offering the flexibility required by specific applications. The J2EE compatibility test suite assures that J2EE platforms, expected to come from numerous vendors, offer a standard level of services and APIs for deploying J2EE-compliant applications. This allows application vendors to create a single application package for all J2EE platforms. Similarly, J2EE platform vendors benefit from a greatly expanded set of applications since compliant applications are guaranteed to run on their product. Platform vendors, however, can differentiate based on the level of security, reliability, scalability, tools support, legacy integration, manageability and other factors transparent to the application. These characteristics give application vendors, platform vendors and customers a great deal of freedom of choice and lead to lower costs for J2EE platforms and applications.

You've seen the promise of J2EE, but can you buy a J2EE platform and begin developing and deploying J2EE applications? Not yet. But today major industry players sell products, usually called application servers, that exhibit many J2EE characteristics. You can use these application servers to begin leveraging the advantages of J2EE. IBM's WebSphere and BEA's WebLogic are examples of currently available application servers. These companies, and many others including Bluestone, Gemstone, Oracle and the Sun/Netscape alliance, are expected to offer J2EE-compliant platforms as soon as it's feasible. Many of the same vendors and lots of other companies, large and small, are expected to produce platform components, application components and full J2EE applications.

The e-business applications of today and tomorrow require a number of enterprise-class characteristics and the flexibility to support varied networked business-to-business and business-to-consumer situations. J2EE promises the very characteristics needed for inexpensive, rapid development and deployment of e-business applications. J2EE creates an environment for e-business applications that allows everyone – J2EE platform vendors, J2EE application vendors, server manufacturers, the businesses using J2EE platforms and applications and, most important, the end user – to win.

AUTHOR BIO

Greg Flurry, a senior technical staff member with IBM, specializes in applying enterprise-class Java technologies to e-business applications.

flurry@us.ibm.com

```
    session.getValue("customer");
  ViewAccounts viewAccts = new ViewAccounts();
  viewAccts.setCustomerID(customerInfo.getID());
  try {
    viewAccts = (ViewAccounts)
      CommandManager.execute(viewAccts);
    Vector accounts = viewAccts.getAccounts();
%>
<!-- NOTE: HTML table setup code deleted -->
<%
    for (int i = 0; i < accounts.size(); i ++) {
      AccountDataBean account =
        (AccountDataBean) accounts.elementAt(i);
      double balance = account.getBalance();
      String formattedBalance =
            currencyFormat.format(balance,
              new StringBuffer(),
              fieldPosition).toString();
%>
    <!-- some formating code deleted -->
    <TR>
    <TD> <%= account.getDescription() %></TD>
    <TD> <%= formattedBalance %></TD>
    </TR>
<%
    }
  } catch (Throwable e) {
    e.printStackTrace();
}
%>
```

Listing 5: The business logic of the ViewAccounts command (ViewAccounts.java)

```
try {
  accounts = new Vector();
  AccountHome accountHome = (AccountHome)
    EntityUtility.lookupHome(AccountHome.class);
    Enumeration accountEn =
      accountHome.findAllAccountsForCustomer
          (customerID);

    while (accountEn.hasMoreElements()) {
      Account account =
        (Account) accountEn.nextElement();
      accounts.addElement(account.getDataBean());
    }

} catch(Exception ex) {
  ex.printStackTrace();
  throw ex;
}
```

Listing 6: Extracts from the Account EJB (AccountBean.java)

```
public void ejbCreate(double balance,
              String description,
              int customerID)
    throws RemoteException, CreateException
{
  this.canWriteChecks = false;
  this.balance = balance;
  this.description = description;
  this.customerID = customerID;
}

public AccountDataBean getDataBean()
    throws RemoteException
{
  AccountDataBean dataBean =
    new AccountDataBean(canWriteChecks,
        balance, description, customerID);
  return dataBean;
}
```

Java Servlet 2.2 Introduces the Web Application

It's literally the tip of the iceberg in building an enterprise application

WRITTEN BY
Arny Epstein

The philosophy behind the Java 2 Enterprise Edition (J2EE) announced at JavaOne in June 1999 is to package the Java 2 platform with a collection of "Enterprise APIs," including Servlet and Java Server Pages (JSP), and an application programming model to define a standard platform upon which enterprise applications can be built. The first public release of the J2EE specification became available in October 1999, and a final draft is expected this month (see http://java.sun.com/j2ee/).

J2EE is a broad specification that covers many of the elements used in building multitier Web-based applications. The end-user–facing tier of a multitier application is the presentation layer. The Servlet and JSP specifications describe a standard methodology for building dynamic HTML or XML pages. Other J2EE technologies, such as JDBC and EJB, describe standard mechanisms for accessing data and business logic. Since the announcement of J2EE, the Servlet specification has been revised once (the current version is Servlet 2.1) and another version is in the process of being reviewed. (Servlet 2.2 is currently available for public review at http://java.sun.com/products/servlet/.)

The broad enterprise application view taken by J2EE has driven extensions and enhancements to the Servlet and JSP specifications. The most important of these is the introduction of the "Web Application" concept, which recognizes that it's necessary but not sufficient to describe how a single servlet should operate. It's also important to describe how all components of an overall application relate to each other.

A typical Web application comprises a collection of static content (e.g., HTML pages, sounds and images) and a collection of dynamically generated pages and images. A Web application also requires management of security, state information in various scopes (e.g., application-wide, per-client session and per servlet) and the isolation of applications and their resources from each other within a server. By taking this broad view of an application, the Servlet 2.2 specification provides a complete programming model for dynamic-content elements within the context of an entire application.

Java Server Pages are a complement to servlets. The goal of JSP is to separate the visual aspects of a Web page from the programmatic aspects. A Web-page designer can focus on laying out the most beautiful HTML and simply use custom HTML tags to identify the areas of dynamic content. At the same time, a professional Java programmer can write the actual code and business logic that's executed to provide the dynamic content to the page. The current version of the JSP specification is 1.0, and a public draft for the 1.1 specification is now available. The remainder of this article will focus on the Web application model as introduced in Servlet 2.2. For more information on JSP see http://java.sun.com/products/jsp/.

Web Application Development

From the developer's point of view, a Web application consists of the following components:

- Servlets and JSPs for generating dynamic content
- Utility Java classes used by servlets and JSPs
- Static documents (HTML pages, images, sounds, etc.) that may be directly URL-accessible and served directly by the Web server (alternatively, these documents may be accessed as part of a servlet's dynamic processing)
- Client-side Java applets, JavaBeans and utility classes (not discussed in this article)
- An XML application descriptor file

These components are packaged in an archive file called a .WAR file (Web Archive).

Web Application Deployment

A Web application (packaged in its .WAR file) is deployed to a "Web Container," available in many Web server products as well as in all J2EE-based Web Application Servers. The Web container provides the environment in which a servlet operates. Upon deployment it's responsible for appropriately adding the contents of the .WAR file to the URL space of its HTTP server.

The XML descriptor file contains the configuration and deployment information that makes this possible. The descriptor file includes information for design tools, deployment tools and details used at runtime by the Web container, including security information.

Security

Consistent with the EJB specification, a Web application can define a set of logical security roles that are enumerated as part of the descriptor file. They're used to control access to any of the servlets in the application as well as any of the static resources. In addition, a servlet may dynamically check on whether the current user is in a particular role.

When the Web application is being deployed, the roles defined in the application must be mapped to actual users and groups.

URL Mapping

The descriptor file enumerates the servlets that are part of the application. For each servlet there's a mapping from one or more URL patterns to that servlet. For example, the following excerpt defines a servlet named "catalog" and the fully qualified name of the class that should be loaded. All other references to the servlet use the name.

```
<servlet>
 <servlet-name>catalog</servlet-name>
 <servlet-class>com.mycorp.Cata-
logServlet</servlet-class>
</servlet>
```

The following excerpt specifies that for the servlet named *catalog* any

request to a URL that begins with "/catalog/" should be delivered to this servlet. In other words, the servlet is written to make /catalog appear to be a directory hierarchy.

```
<servlet-mapping>
 <servlet-name>catalog</servlet-name>
 <url-pattern>/catalog/*</url-pat-
tern>
</servlet-mapping>
```

Web Application Runtime

The Web container is responsible for managing the runtime behavior of the components of the Web application (see Figure 1). The servlet API defines the contract between the Web container and the servlet objects that are deployed in it. The Web container must:

- Provide an application-wide context object
- Map URL requests within its root URL space according to the application descriptor information
- Create and manage client sessions
- Directly deliver URL-accessible static content
- Invoke servlets to deliver dynamic content

JAVAX.SERVLET.SERVLETCONTEXT

To represent the running Web application, the servlet container creates an instance of javax.servlet.ServletContext. This ServletContext object, available to each servlet within the application, provides three functions.

1. It gives a servlet access to other resources (HTML pages, images, etc.) within the application's .WAR file via the getResource and getResourceAsStream methods.
2. It's the manager of application-wide state information via the get/setAttribute methods.
3. It's a factory for javax.servlet.RequestDispatcher objects. The RequestDispatcher is used by one servlet to invoke another servlet from within the same application. This allows a servlet to delegate its processing to a different servlet (forwarding) or to ask another servlet to provide a piece of the response (inclusion).

JAVAX.SERVLET.HTTP.HTTPSESSION

To represent a client HTTP session, the Web container creates an instance of javax.servlet.http.HttpSession. This object is used by a servlet to store state information that's per-client. The Servlet specification doesn't explain how a Web container should identify a session. Two common mechanisms are used for session identification: cookie-based and URL rewriting (cookieless). In addition, HTTP over SSL (HTTPS protocol) is inherently session based. In the case of an HTTPS session, the Web container makes the X509 certificate information available as a Request attribute.

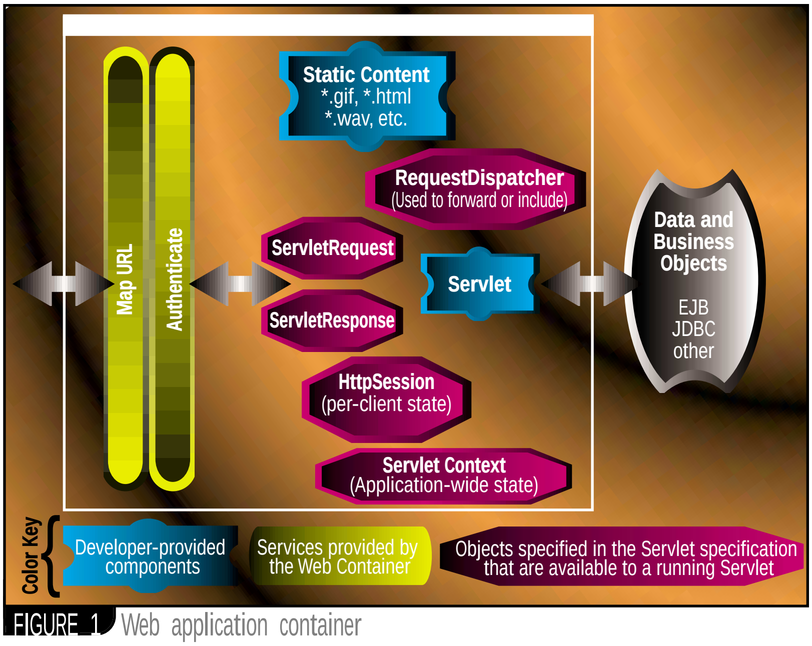

Static Content
*.gif, *.html
*.wav, etc.
RequestDispatcher
(Used to forward or include)
Data and Business Objects
EJB
JDBC
other
Map URL
Authenticate
ServletRequest
Servlet
ServletResponse
HttpSession
(per-client state)
Servlet Context
(Application-wide state)
Color Key
Developer-provided components
Services provided by the Web Container
Objects specified in the Servlet specification that are available to a running Servlet

FIGURE 1 Web application container

JAVAX.SERVLET.SERVLET

A servlet is a Java class that implements the javax.servlet.Servlet interface. The purpose of a servlet is to generate dynamic content. It interacts with clients using a Web-style request-response mechanism based on the behavior of the HTTP protocol. (*Note*: The Servlet specification is protocol independent, but at least the HTTP protocol is required. For the purposes of this article, HTTP will be assumed.) At runtime a JSP behaves exactly like a servlet. In fact, most JSP hosts precompile a JSP into a compiled servlet. Throughout this article, wherever the term *servlet* is used, the phrase *servlet or JSP* can be substituted.

When a servlet is selected to process a request, its "service" method is invoked by the RequestDispatcher. The service method accepts two parameters: a Request object and a Response object. The servlet uses the information provided in the Request object and the ServletContext to determine what dynamic content is to be generated.

JAVAX.SERVLET.SERVLETREQUEST

When a client request received by the server requests a servlet, an instance of the servlet is passed an object that implements the javax.servlet.ServletRequest interface. For HTTP-based clients, the object must implement javax.servlet.http.HttpServletRequest. The Request object provides all the information delivered from the HTTP client including parameters, headers, path elements and attributes.

Request parameters are obtained from the request query string (following the "?") and from HTML form data (in the case of an HTTP POST). Parameter information is provided by the end user or the application itself. For example, the request string www.mysite.com/here/there/now.html?a=10&b=no-way specifies two request parameters with the names "a" and "b" and corresponding values of "10" and "no-way."

Request headers, a standard part of the HTTP and other Internet protocols, are generated by the HTTP client program or user agent, usually a browser or client-side Java program. An example of a request header is "User-Agent." The value of this header for Netscape Navigator 4.5 is "Mozilla/4.6 [en] (WinNT; I)". To properly use request headers, the programmer must have a working knowledge of the HTTP 1.1 protocol.

Request-path elements are pieces of the entire request path broken down functionally (from left to right) into three parts. First is the "Context Path," which is the path to the root URL of the Web application. Second is the path from the root URL of the Web application to the servlet itself. (Mapping of URLs to servlets is defined by the application descriptor file.) The third part of the path is the "Extra Path Info" that's interpreted by the servlet.

For example, a .WAR file is deployed as follows:

- *Application root*: "/here"
- The *application descriptor* maps the path "/there/fancy.blah" to the servlet "MyServlet"

A request for "/here/there/fancy.blah/more/and/more" then delivers to "MyServlet" the following three path elements:

1. *Context path*: "/here"
2. *Servlet path*: "/there/fancy.blah"
3. *Extra path*: "/more/and/more"

Attributes are information represented as name-value pairs and are internal to the application. A typical use for Request attributes is to pass parameters from one servlet to another when a request is being forwarded.

JAVAX.SERVLET.SERVLETRESPONSE

The response object, which implements the javax.servlet.ServletResponse interface, is used to encapsulate all information to be returned to the client. The Request object allows the servlet to set the return status (e.g., 200 OK, 404 Not Found, 304 Not Modified, etc.), response headers and the actual content (HTML, text, image, etc.).

New in Servlet 2.2, the Response object has additional methods that enable the servlet to control buffering of the response. This enhancement will greatly improve performance and reduce heap usage by enabling a servlet to indicate that it knows the size of its content and doesn't need any buffering.

SERVLET REQUEST PROCESSING

The real work of a servlet, of course, is to process a client request and generate dynamic content. The servlet specification includes three features that enable a flexible component-based implementation of an application.

FORWARD

A servlet may forward-process a request to another servlet. This capability can be accomplished using HTTP redirection, but forwarding is much more efficient because the network round-trip is avoided and Java objects may be passed between the servlets as Request attributes (avoiding the need to encode complex state information).

A useful application of forwarding is to report errors detected during servlet processing.

INCLUDE

A servlet may include the generated output of another servlet. This is useful for common design elements like navigation bars or for encapsulating the complex generation of table-based results that may be displayed in many page contexts.

ERROR HANDLING

Good error handling and reporting are important to building a user-friendly application. A Web application has an error-handling page that's invoked when an uncaught exception is thrown during servlet processing. This page ensures that the Web container will recover gracefully from unexpected application errors. The Web container provides an error-handling page by default, but the application can specify error pages to handle specific HTTP errors or Java exceptions.

Conclusion

In this article we've provided an overview of the Web application architecture as introduced by the upcoming Servlet 2.2 specification. Driven by J2EE, the Servlet and JSP-based Web application is literally the tip of the iceberg in building an enterprise application. It's the part you see, but there's much more underneath that's seamlessly integrated. The rest of the J2EE technologies, such as JDBC, JTA, JMS, EJB and JavaMail, are the workhorses that provide the dynamic data and business processes that a Web application presents to the user. The Servlet 2.2 specification is a giant step in the creation of a well-integrated, top-to-bottom platform for building enterprise applications.

AUTHOR BIO

Arny Epstein, chief technology officer and a cofounder of SilverStream, holds a Ph.D. in astrophysics from MIT. While at the Harvard-Smithsonian Center for Astrophysics, Arny contributed to the creation of data-analysis techniques in the field of experimental X-ray astronomy.

aepstein@silverstream.com

Fall Internet World

A report on the trade show held recently in New York City

WRITTEN BY
JIM MILBERY

There's nothing better than a technology trade show in New York City in the fall. While leaves may be gently falling from the trees in New England, the movers and shakers of the Internet gather like so many Gordon Geckos to plot strategy, change the world and otherwise meet and greet the masses in the city that never sleeps.

PHOTO 1 The green-screen registration area

Having spent years as a sales consultant for a variety of high-tech vendors, I always feel a slight trepidation when I cross the threshold onto the exhibitor's floor. For a sales consultant the trade show floor is one long line of unqualified prospects filling their trick-or-treat floorshow bags with giveaway goodies and looking for a quick demo. The booth is either filled with eager faces or a ghost town – there's no middle ground. Attending one of these things as a civilian is a lot more fun, take my word for it. Thanks to ***JDJ***, I was able to skirt the long line at the day-pass registration booth using my VIP pass. Like a celebrity hopping the line at the old Studio 54, I was invited to duck under the velvet rope and make my way to the registration area without waiting.

The registration process itself was very disappointing. I like the idea of self-registration, but asking attendees of an Internet show to fill out registration forms using green-screen terminals is ridiculous (see Photo 1). Don't get me wrong. I'm not advocating the destruction of terminals, and there are applications in which terminals still make sense. It's just that a trade show for the Internet is the wrong venue for them. While I'm sure the registration people are armed with statistics about how efficient the registration process was (blah, blah, blah), most of the people I spoke with were very frustrated by the terminal-based data-entry screens. Forget the fact that most of the registration process consisted of answering a series of self-serving demographic questions, the application itself was difficult to use. In the age of the Internet many people aren't familiar with terminal-based applications – and I find it hard to believe that a few PC vendors couldn't be strong-armed into providing enough computers for browser-based registration. Many of the end users I talked to had never even seen a nondestructive delete-key in action, and they had lots of trouble with the application as a result. (By the way, guys, frustrated users rarely give you accurate answers to all those pithy demographic questions.) Internet World has grown into a conference for all Web constituents, not just developers. If you want businesspeople and end users to attend this type of event, you can't ask them to use technology that's completely unfamiliar to them when they register.

Automated registration processing is a good thing and I'm sure it saves people a ton of time. At a show like Internet World, however, the registration application should be browser-based and come equipped with adequate help and automated assistance.

A "United Nations" of Technology

Despite the hiccups with the registration process, the show itself was terrific. While Comdex may have gotten out of hand, Internet World is just the right size – despite the fact that the exhibition area itself is split between two floors. The show is truly a United Nations of technology, with representatives from all facets of computing and the Internet. While the keynote speeches and breakout sessions tend to capture most of the press, the exhibition floor is where the real action is, with everything you need to conceive and build a Web site or Internet application.

While it might be difficult for the novice to figure out all the pieces of the puzzle without some help, the fact remains that everything you need lies right there in the hall of vendors. What is truly incredible is the vast array of different technologies gathered under one roof. From telecom to online banking and from computer games to professional liability insurance for Web developers – Internet World has it all. While shows like JavaOne provide developers with a focused environment for studying and discussing technology, Internet World provides a more global and holistic approach to the Internet in general.

One of the continuing mantras for developers and information technology staff is that they should keep close with the needs of the end user. There's no bet-

PHOTO 2 Santa's Internet workshop

ter place for understanding how users are viewing and interacting with technology and the Internet than this forum. Exposing developers to the complete view, including end-user applications of Java technology, reap untold benefits in terms of mutual understanding.

Even with the increased focus on such things as e-commerce and customer-resource management, it's clear that the critical item on everyone's agenda is content and content management. Many of the products I looked at were geared toward handling content and getting information published into Web portals – a key issue for corporations and dotcom companies alike. Portal-based applications allow corporations to get information disseminated among their many employees and divisions and customers courtesy of intranets and extranets. Although we have a history in the IT business of jumping onto too many bandwagons as the "next big thing," this seems to be one strategy with some real legs to it. For the dotcom set, portal-based applications are the lifeblood of their business. Organizing and personalizing content means the difference between winning and losing for these companies. What's really interesting is that a show like Internet World can service both constituencies so well. I'd have no trouble crafting together a complete solution for the corporate marketplace or the dotcom marketplace with the vast array of products on display there. Everything from bandwidth and server hosting to customer service applications were to be found on the show floor. While many of these solutions are Java-based products, the emphasis at this show was less on the underlying technologies than on solving the bigger problem. Java Server Pages and XML are what Web developers are talking about these days, but the focus of Internet World was the end result of using these technologies, rather than the technology itself.

Some Specialties

While it's not my place to hand out awards and endorse products, I do have to mention a few things that caught my eye. First off, you have to love the idea of having a Christmas-themed booth complete with packs of roving Santa Clauses (all of whom had New York accents) at a technology trade show (see Photo 2). One of the great things about the Internet is the way it has fused the interests of stuffy corporations, technical developers and creative types.

Take Abject Modernity as an example. Buried among the telecom providers, development tools and CRM applications was this clever little company out of Canada with their intriguing game "The Stone." The Stone itself is a sort of game device that you use to solve a series of puzzles in the grand hope of learning the secret of the Enigma. Once you buy your Stone, you can register it on the Web site – provided that you can figure out the secret registration code hidden in the booklet. Now that's just plain fun, and it's nice to see that while the Internet is growing up into serious business, we can still have fun with it. If you want the scoop on technical Java topics go to JavaOne or the Java Business Conference – but Internet World remains the one spot where you can find everything else under the Internet sun.

jmilbery@kuromaku.com

AUTHOR BIO

Jim Milbery is an independent software consultant based in Easton, Pennsylvania. He has over 15 years of experience in application development and relational databases. You can visit Jim's Web site at www.kuromaku.com.

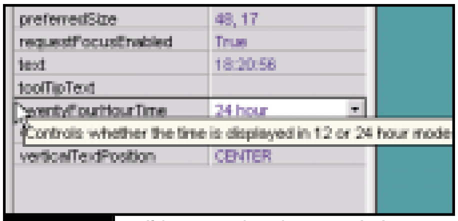
JavaBeans Customized...

continued from page 12

that are not bean properties. The “name” is the name that JBuilder will give the Clock variable in the code that it generates. “Constraints” refers to the layout constraints used when the Clock component is added to its container.

The custom ClockCustomizer looks a lot like it did in the BeanBox. It’s displayed by right-clicking on the Clock component and selecting Customizer from the popup menu. JBuilder’s version of the ClockCustomizer is shown in Figure 8.

JBuilder also has a design view that shows the user interface component tree. In this view JBuilder shows the Clock bean contained within a JPanel (see Figure 9). Here it used the 16x16 color icon that I specified in ClockBeanInfo.



preferredSize	48, 17
requestFocusEnabled	True
text	18:20:56
toolTipText	
twentyFourHourTime	24 hour
verticalTextPosition	CENTER

Controls whether the time is displayed in 12 or 24 hour mode

FIGURE 7 JBuilder uses the ShortDescription text.

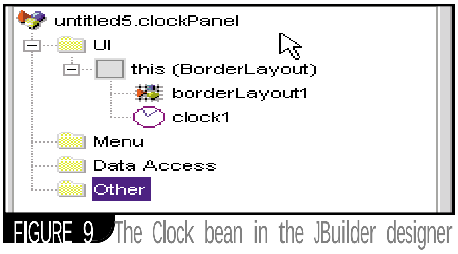


FIGURE 9 The Clock bean in the JBuilder designer

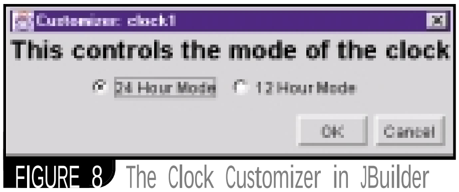


FIGURE 8 The Clock Customizer in JBuilder

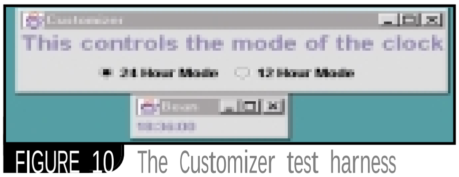


FIGURE 10 The Customizer test harness

Debugging a Customizer

This example is a simple customizer, but for a complicated customizer it might be necessary to use a debugger to get it working just right. Once the bean and customizer are loaded into an IDE, it can be pretty difficult to find and fix problems. Sometimes even the time-tested method of adding System.out messages won’t work because there is no console to print to. I’ve found that it’s worth taking the time to write code that can be used to test customizers outside of an IDE. That way I can use System.out or a debugger to see what the customizer is doing. This simple code loads a bean and its customizer into two windows so they can be tested and debugged.

```
Clock c = new Clock();
JFrame f = new JFrame(“Bean”);
f.getContentPane().add(c);
f.pack();
f.show();

ClockCustomizer cust = new ClockCustomiz-
er();
f = new JFrame(“Customizer");
f.getContentPane().add(cust);
f.pack();
f.show();
cust.setObject(c);
```

Figure 10 shows what the test harness looks like.

This simple test harness can make debugging a customizer much easier. Insert this code into the main() method of the customizer for a built-in test capability. With a little additional code this test driver could also be used to test bean serialization and event handling.

Using custom property editors and customizers are great ways to make your beans more usable by application developers regardless of what development tools they use. Take the time to include these features in your beans and your users will thank you for it.

AUTHOR BIO

William Wright is a senior software engineer with GTE Corporation in Arlington, Virginia. He has 10 years' experience with real-time systems development and object-oriented programming.

wwright@bbn.com

Listing 1: The Clock Bean

```
package mybeans;

import java.awt.*;
import java.util.*;
import java.text.*;
import javax.swing.*;
import java.io.*;

public class Clock extends JLabel
     implements Runnable, Serializable{
  DateFormat formatter12;
  DateFormat formatter24;
  private boolean running;
  private boolean isTwentyFourHourTime;

  public Clock() {
    formatter12 =
     new SimpleDateFormat("h:mm:ss a");
    formatter24 =
         new
SimpleDateFormat("H:mm:ss");
    setRunning(true);
  }

  public void run() {
  while(running)
  {
   if (isTwentyFourHourTime())
    this.setText(
     formatter24.format(new Date()));
    else
     this.setText(
      formatter12.format(new Date()));
     try {
       Thread.sleep(1000);
     }
     catch (InterruptedException e) {}
   }
}

// The running property should be hidden
public void setRunning(boolean newRun-
ning) {
  boolean  oldRunning = running;
  running = newRunning;
  Thread t = new Thread(this);
  t.start();
  firePropertyChange("running",
             new Boolean(oldRunning),
             new Boolean(newRunning));
}

public boolean isRunning() {
  return running;
}

 // The twentyFourHourTime property
 // should be exposed
 public void setTwentyFourHourTime(
       boolean newTwentyFourHourTime) {
 boolean  oldTwentyFourHourTime =
       isTwentyFourHourTime;
 isTwentyFourHourTime = newTwenty-
 FourHourTime;
 firePropertyChange("twentyFourHourTime",
    new Boolean(oldTwentyFourHourTime),
    new Boolean(newTwentyFourHourTime));
 }

 public boolean isTwentyFourHourTime() {
  return isTwentyFourHourTime;
 }
}
```

Listing 2: The Property Editor

```
package mybeans;

import java.beans.*;

public class ClockPropertyEditor
  extends PropertyEditorSupport {

 private boolean is_24 = false;

 public ClockPropertyEditor() {
 }

 public String getAsText() {
   return (is_24 ? "24 hour" : "12
   hour");
 }

 public Object getValue() {
   return new Boolean(is_24);
 }

 public void setAsText(String value)
        throws IllegalArgumentExcep-
tion {
   if (value.equals("24 hour"))
     setValue(new Boolean(true));
   else if (value.equals("12 hour"))
     setValue(new Boolean(false));
   else
     throw new IllegalArgumentExcep-
tion(
                "Unrecognized value: "
+ value);
 }

 public void setValue(Object value) {
   Boolean b = (Boolean)value;
   is_24 = b.booleanValue();
   firePropertyChange();
 }

 public String[] getTags() {
   String [] tags = {"24 hour", "12
hour"};
   return tags;
 }
}
```

Listing 3: The ClockBeanInfo Class

```
package mybeans;

import java.beans.*;

public class ClockBeanInfo
       extends SimpleBeanInfo {
  public ClockBeanInfo() {
  }

  public
  PropertyDescriptor[] getPropertyDescriptors() {
    try  {
      // PropertyDescriptor for the
      // "twentyFourHourTime" property
      PropertyDescriptor pd1 =
        new PropertyDescriptor(
            "twentyFourHourTime",
             Clock.class,
            "isTwentyFourHourTime",
            "setTwentyFourHourTime");
      pd1.setDisplayName("Time Format");
      pd1.setShortDescription(
        "Controls whether the time is "+
        "displayed in 12 or 24 hour mode");
      pd1.setBound(true);
      pd1.setPropertyEditorClass(
        ClockPropertyEditor.class);

      // PropertyDescriptor to hide the
      // "running" property
      PropertyDescriptor pd2 =
        new PropertyDescriptor(
            "running",
            Clock.class,
            "isRunning",
            "setRunning");
      pd2.setHidden(true);

      PropertyDescriptor[] pds = new PropertyDescriptor[]
{pd1, pd2};
      return pds;
    }
    catch(IntrospectionException ex) {
      return null;
    }
  }

  public BeanDescriptor getBeanDescriptor() {
    return new BeanDescriptor(
      Clock.class,
      ClockCustomizer.class);
  }

  public java.awt.Image getIcon(int iconKind) {
    switch (iconKind) {
      case BeanInfo.ICON_COLOR_16x16:
        return loadImage("Clock16x16Color.gif");
      case BeanInfo.ICON_COLOR_32x32:
        return loadImage("Clock32x32Color.gif");
      case BeanInfo.ICON_MONO_16x16:
        return loadImage("Clock16x16Mono.gif");
      case BeanInfo.ICON_MONO_32x32:
        return loadImage("Clock32x32Mono.gif");
    }
    return null;
  }

  public BeanInfo[] getAdditionalBeanInfo() {
    Class superclass = Clock.class.getSuperclass();
    try  {
      BeanInfo superBeanInfo =
Introspector.getBeanInfo(superclass);
      return new BeanInfo[] { superBeanInfo };
    }
    catch(IntrospectionException ex) {
      return null;
    }
  }
}
```

Listing 4: The Clock Customizer

```
package mybeans;

import java.awt.*;
import java.beans.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;

public class ClockCustomizer extends JPanel
                      implements Customizer{
  protected Clock theClockBean = null;
  private JRadioButton button24 =
      new JRadioButton("24 Hour Mode");
  private JRadioButton button12 =
      new JRadioButton("12 Hour Mode");

  public ClockCustomizer() {
    JLabel topLabel = new JLabel();
    topLabel.setFont(
      new java.awt.Font("Dialog", 1, 20));
    topLabel.setHorizontalAlignment(
      SwingConstants.CENTER);
    topLabel.setText(
      "This controls the mode of the clock");
    this.setLayout(new BorderLayout());
    button24.addChangeListener(
      new ChangeListener() {
        public void stateChanged(ChangeEvent e) {
          button24Changed(e);}});
    this.add(topLabel, BorderLayout.NORTH);
    JPanel radioPanel = new JPanel();
    this.add(radioPanel, BorderLayout.CENTER);
    radioPanel.add(button24, null);
    radioPanel.add(button12, null);
    ButtonGroup grp = new ButtonGroup();
    grp.add(button12);
    grp.add(button24);
    button12.setSelected(true);
  }

  public void setObject(Object bean) {
    theClockBean = (Clock)bean;
    button24.setSelected(
      theClockBean.isTwentyFourHourTime());
  }

  void button24Changed(ChangeEvent e) {
    theClockBean.setTwentyFourHourTime(
      button24.isSelected());
  }
}
```

Download the Code!

The code listing for this article can also be located at

www.JavaDevelopersJournal.com

JDJ NEWS

ObjectSpace Releases DXML

(*Dallas, TX*) – ObjectSpace has announced the availability of Dynamic XML 1.0 for Java, which is said to simplify XML development by allowing developers to create, write and read XML documents as if they were standard JavaBeans. DXML is available free to Java developers.
www.objectspace.com

JRun to Power In-Store Kiosks for Home Depot

(*Cambridge, MA*) – Home Depot, the huge home improvement retailer, has chosen Allaire JRun to power its in-store information kiosks.

According to Ron Griffin, CIO of Home Depot, "We were looking for a solution that would enable us to deliver product and vendor information quickly and efficiently to our customers....JRun provides the most flexibility and performance. And it enables us to maximize our physical 'floor space' while providing robust and easy-to-use information kiosks for our customers."
www.allaire.com

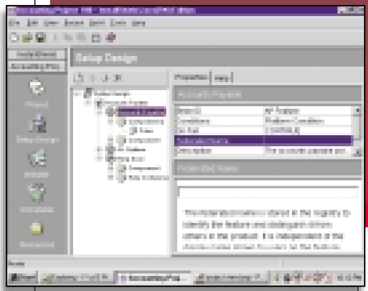
New Vice President at Insignia Solutions

(*Fremont, CA*) – Mark McMillan has joined Insignia Solutions as senior vice president of worldwide sales and marketing. McMillan comes to Insignia from Phoenix Technologies, where he was VP of sales for the company's Internet division.

In his new role McMillan will be responsible for managing Insignia's worldwide sales and marketing organization, including direct and OEM sales, product and sales strategy, positioning, business development, product marketing and communications.
www.insignia.com

InstallShield Introduces Java Edition 3.0

(*Schaumburg, IL*) – InstallShield Software Corporation has released InstallShield Java Edition (ISJE) 3.0, the newest version of its true cross-platform installation authoring tool for multiplatform software developers. ISJE's new features and robust functionality claim to give developers increased control and customization over their installations while providing an intuitive "Windows-like" installation experience, thus reducing confusion and the learning curve often associated with installing non-Windows software applications. ISJE 3.0 is also said to enhance installation and deployment to the Solaris operating environment by leveraging Web Start Wizard technology jointly developed by InstallShield and Sun.
www.installshield.com



IBM Names Java VP

(*White Plains, NY*) – IBM has selected Rod Smith as its new vice president for Java. He replaces Pat Sueltz, who joined Sun Microsystems in September.

Smith was IBM's chief technology officer for Java and has worked closely with Sun, especially on embedded Java, for which standards aren't yet established. Smith's appointment is expected to be good news for Java because Sueltz and Smith have worked well together in the past. IBM has played a pivotal role in supporting Java while at the same time pushing Sun to keep Java open. Smith said his priorities are to "keep IBM's full focus on Java going and to keep the team together."
www.ibm.com

TOPLink for BEA WebLogic Server Launched

(*San Jose, CA*) – BEA Systems Inc. and The Object People have launched TOPLink for BEA WebLogic Server. Customers building e-commerce applications using enterprise Java standards can now map and transparently store Enterprise JavaBeans in object relational databases, thereby reducing application development time by 40% or more.
www.beasys.com / www.objectpeople.com

PointBase to Bundle Database with Symantec's VisualCafé

(*San Mateo, CA*) – PointBase, Inc., will bundle the PointBase 100% Pure Java relational database with future releases of Symantec's VisualCafé. VisualCafé applications written with PointBase's standards-based Java relational database can now be migrated to any corporate database system that supports both SQL and JDBC.
www.pointbase.com / www.symantec.com

ADVERTISING INDEX

ADVERTISER	URL	PH	PG
4TH PASS	WWW.4THPASS.COM	877.484.7277	17
9NETAVENUE, INC.	WWW.9NETAVE.NET	888.9NETAVE	79
AFFINIA	WWW.AFFINIA.COM	650.404.9944	101
AMERICAN CYBERNETICS	WWW.SOFTEXPORT.COM	800.899.0100	43
APPLIED REASONING	WWW.APPLIEDREASONING.COM	800.260.2772	35
BEA WEBLOGIC	WWW.WEBLOGIC.BEASYS.COM	800.817.4BEA	2
BLUE SKY SOFTWARE	WWW.BLUE-SKY.COM	800.559.4423	15
CAREER CENTRAL	WWW.CAREERCENTRAL.COM/JAVA	888.946.3822	44
CAREER OPPORTUNITY ADVERTISERS		800.846.7591	87-100
CERTIFY ON-LINE	WWW.CERTIFYONLINE.COM	877.JAVA YES	16
COMPUWARE NUMEGA	WWW.COMPUWARE.CON/NUMEGA	800.4.NUMEGA	6
CYSCAPE	WWW.CYSCAPE.COM/FREE4J	800.932.6869	58
DEVELOPMENTOR	WWW.DEVELOP.COM	800.699.1932	85
ELIXIR TECHNOLOGY	WWW.ELIXIRTECH.COM/DOWNLOAD/	65 532.4300	71
ENTERPRISESOFT	WWW.ENTERPRISESOFT.COM	510.742.6700	11
FIORANO SOFTWARE, INC.	WWW.FIORANO.COM	408.354.3210	39
GEEK CRUISES	WWW.GEEKCRUISES.COM	650.327.3692	82
IAM CONSULTING	WWW.IAMX.COM	212.580.2700	59
INETSOFT TECHNOLOGY CORP	WWW.INETSOFTCORP.COM	732.235.0137	57
INSIGNIA SOLUTIONS, INC.	WWW.INSIGNIA.COM	800.848.7677	63
INSTANTDB	WWW.INSTANTDB.CO.UK		74
INSTANTIATIONS INC.	WWW.INSTANTIATIONS.COM	800.808.3737	64
INTUITIVE SYSTEMS, INC.	WWW.OPTIMIZEIT.COM	408.245.8540	55
JAVA BUSINESS CONFERENCE	WWW.JAVABUSINESSCONFERENCE.COM	888.886.8309	77
JAVA DEVELOPER'S JOURNAL	WWW.JAVADEVELOPERSJOURNAL.COM	914.735.0300	86
JDJ STORE	WWW.JDJSTORE.COM	888.303.JAVA	83
KL GROUP INC.	WWW.KLGROUP.COM/TRACK	888.328.9597	13
KL GROUP INC.	WWW.KLGROUP.COM/PAGELAYOUT	888.328.9599	69
KL GROUP INC.	WWW.KLGROUP.COM/FIELD	888.328.9596	104
METAMATA, INC.	WWW.METAMATA.COM	510.796.0915	51
NEW ATLANTA	WWW.NEWATLANTA.COM/	678.366.3211	47
OBJECT DESIGN	WWW.OBJECTDESIGN.COM/JAVLIN	800.962.9620	52-53
OBJECTSWITCH CORPORATION	WWW.OBJECTSWITCH.COM/IDC35/	415.925.3460	29
POINTBASE	WWW.POINTBASE.COM/JDJ	877.238.8798	41
PRAMATI	WWW.PRAMATI.COM/J2EE.HTM	914.876.3007	23
PROTOVIEW	WWW.PROTOVIEW.COM	800.231.8588	3
PROTOVIEW	WWW.PROTOVIEW.COM	800.231.8588	75
QUICKSTREAM SOFTWARE	WWW.QUICKSTREAM.COM	888.769.9898	30
RIVERTON SOFTWARE CORPORATION	WWW.RIVERTON.COM	781.229.0070	73
SEGUE SOFTWARE	WWW.SEGUE.COM/ADS/CORBA	800.287.1329	27
SIC CORPORATION	WWW.ACCESS21.CO.KR	822.227.398801	21
SILVERSTREAM SOFTWARE, INC.	WWW.SILVERSTREAM.COM	888.823.9700	103
SLANGSOFT	WWW.SLANGSOFT.COM	972.375.18127	38
SLANGSOFT	WWW.SLANGSOFT.COM	972.375.18127	78
SD '00 WEST	WWW.SDEXPO.COM		65
SOFTWARE INSTRUMENTS	WWW.SO-IN.COM	972.633.8555	81
SOFTWIRED INC.	WWW.JAVAMESSAGING.COM/IBUS	(41) 1.445.2370	7
SYBASE INC.	WWW.SYBASE.COM	800.8.SYBASE	25
SYMANTEC	WWW.SYMANTEC.COM		4
SYS-CON PUBLICAITONS	WWW.SYS-CON.COM	800.513.7111	84
TIDESTONE TECHNOLOGIES	WWW.TIDESTONE.COM	800.884.8665	45
TOGETHERSOFT LLC	WWW.TOGETHERSOFT.COM	919.772.9350	49
UNIFY CORPORATION	WWW.EWAVECOMMERCE.COM	800.GO.UNIFY	37
VISICOMP, INC.	WWW.VISICOMP.COM	831.335.1820	31
VISUALIZE INC.	WWW.VISUALIZEINC.COM	602.861.0999	85
VSI	WWW.VSI.COM/BREEZE	800.556.4VSI	33

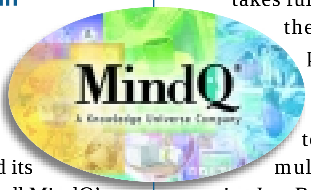
JDJ NEWS

MERANT Supports J2EE

(*Mountain View, CA*) – MERANT announces support for Java Database Connectivity 2.0 technology features essential for building applications based on Java 2 Platform, Enterprise Edition. Extending data access and integration capabilities across enterprise-class Java applications, the JDBC 2.0 technology is said to provide customers with increased interoperability, flexibility and performance. The new JDBC 2.0 technology support is included in an upgraded version of MERANT's DataDirect SequeLink product, currently in customer beta trials, and will be widely available in early 2000.

www.merant.com

Oracle and MindQ Team Up Again



(*Reston, VA*) – MindQ Publishing, Inc., announces that Oracle Corporation has renewed its agreement to resell MindQ's developer training for Java. In addition, MindQ will be developing four custom Java courses for Oracle to include in their portfolio. The two companies also plan to work together to help developers fulfill the requirements of the Certification Initiative for Enterprise Development, an industry-wide effort to establish standards for skill validation of Java developers.

www.mindq.com

Symantec Delivers VisualCafé 4

(*Cupertino, CA*) – Symantec Corporation has released the VisualCafé 4 development environment for Java. The next-generation VisualCafé takes full advantage of the new Java 2 platform standards from Sun Microsystems, including multiserver Enterprise JavaBeans support, Java ServerPages, servlets, CORBA, multitier distributed debugging and other advanced Java technologies.

www.symantec.com

Pervasive Outlines Strategic Initiatives

(*Austin, TX*) – Pervasive Software Inc. has outlined its "Applications Everywhere" strategy designed to further capitalize on growth opportunities in the market for e-business applications. Pervasive is planning significant incremental investments over the next four quarters as it launches an e-business alliance program designed to unite Web application developers with Web integrators, a telemarketing call center to provide sales and technical assistance, a brand awareness campaign and accelerated delivery of new features in Tango 2000 and Pervasive.SQL 2000. In addition, Pervasive has expanded support for corporate servers to include availability of Pervasive.SQL 2000 and Tango 2000 for Solaris and Linux.

www.pervasive.com

Rational Accelerates E-Development

(*Lexington, MA*) – Rational introduces Rational Suite 1.5 and Rational ClearCase 4.0, featuring new integration and enhanced functionality that enable software teams to design, build, test and manage change to Internet software. This integration is facilitated via Unified Change Management (UCM), a new workflow process that combines products to automate the management of activities and software artifacts across the e-development lifecycle.

www.rational.com

StarBase Corporation to Acquire ObjectShare

(*Santa Ana, CA*) – StarBase Corporation, a leading provider of e-business lifecycle management solutions, has signed a definitive agreement to acquire ObjectShare, Inc., a leading provider of object-oriented and e-business professional services.

"This acquisition represents another important step in the execution of our strategy to offer our customers complete end-to-end e-business lifecycle products and services," according to William Stow, CEO of StarBase.

www.starbase.com

Vision Software Closes Mezzanine Funding Round

(*Oakland, CA*) – Vision Software has closed a $15.7 million mezzanine funding round led by Goldman Sachs that also includes investments from Marc Andreessen, founder of Netscape, Inc., and Loewenthal Capital, exclusive investor for Hasso Plattner, cofounder and CEO of SAP, Inc., along with others. Vision Software will leverage the investment to fuel continued adoption of its e-business automation system.

www.vision.com

Employment
Ad

Employment
Ad

Employment
Ad

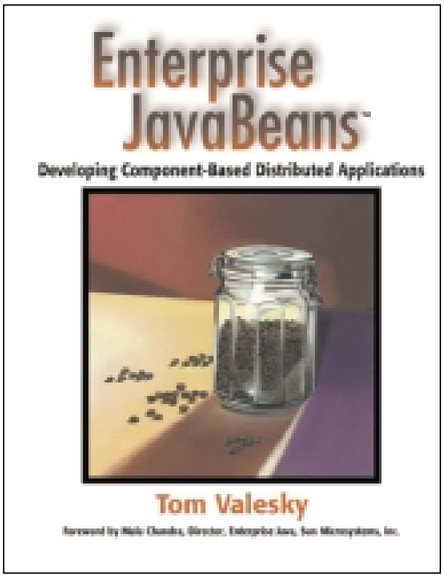
Employment
Ad

An Introduction to EJBs with Lots of Code

Enterprise JavaBeans:
Developing Component-Based Distributed Applications
by Thomas C. Valesky
352 pages, Addison Wesley

REVIEWED BY AJIT SAGAR

This was actually the first book on Enterprise JavaBeans that came into the market. *Enterprise JavaBeans* was released in June and made its debut at JavaOne this year. This is a pretty good book for developers who like to see a lot of code. The examples in the book are used to develop a fairly complex application and the code isn't meant for novices. Tom Valesky presents many examples. I like the fact that the book takes an example and builds its complexity in successive chapters. There's good coverage of distributed architectures and transactions. The author has also dedicated a chapter to provide some excellent guidelines for building distributed systems. For readers just starting out in distributed applications, the author provides the appropriate background. This book isn't for everyone, though. Readers already familiar with distributed systems and transactions may not want to go through the tutorial style followed in the book and may find the detailed code discussions irritating.



The book is written in an easy, informal, tutorial style. It's well organized and easy to follow. In some places the discussion becomes pretty terse, so you may want to take a few breaks when reading. My suggestion to readers who go out and purchase this book: get the software for running the code, install it and *then* read the book. Test out the examples and read the book in sequence.

Chapters 1 and 2 focus on introducing the reader to the concepts that form the basis of this book. The first chapter provides an excellent discussion on the evolution of computer architectures from the single-tier mainframe model to the *n*-tier distributed architecture. In this discussion the author also covers the basics of distributed transaction processing. The chapter ends by showing how Enterprise JavaBeans fit into the big picture. A brief introduction to the EJB spec is provided. Chapter 2 discusses the Enterprise JavaBeans architecture. The concepts are explained well, with the right level of detail. It makes a good stand-alone chapter for reading without requiring the reader to read the rest of the book. The author introduces the appropriate number of class methods to give an overview without confusing the reader. The code is developed using WebLogic and EJBHome software packages.

Chapters 3–6 focus on the various types of EJB development. These chapters follow a pattern: an introduction to the concept, followed by design choice for the example, then a detailed discussion of the example as it is built in parts. It concludes with an analysis of what goes on behind the scenes. As mentioned earlier, a tutorial style is followed, with Chapter 3 focusing primarily on creating a simple client for a "hello, world" example. The design, implementation and deployment of software are explained in detailed, concise steps. The last section of this chapter summarizes what really went on behind the scenes as the reader developed this example. This is again typical of the author's tutorial style. Chapter 4 provides a good overview of session beans. The example in this chapter successfully brings out the main features and functionality offered by session beans. It also discusses transactions vis-à-vis EJBs. An online shopping example is developed using stateful session beans, followed by an example using stateless session beans. The next chapter (Chapter 5) discusses entity beans and persistence management in EJBs. The examples offered in this chapter reuse code from the previous chapter to illustrate design with container-managed persistence as well as bean-managed persistence.

Chapter 6 deals with the topic of developing EJB clients. The author purposely deferred discussions on writing the client till this chapter. Instead of implementing a complicated client, he gives examples of "small" single-purpose clients that bring out different points he's trying to make. This makes the code and the discussion very easy to follow. The example gets a little long, but is appropriate at this stage. Chapter 7 goes into some depth about a few necessary but pretty boring parts of the EJB spec: deployment. The author does a good job of providing the requisite information for the reader to get a handle on this topic. The chapter ends with a discussion on deployment issues that developers may face such as caching and persistence. The author also provides a set of guidelines for developers to follow when dealing with EJB containers.

Chapter 8 is a unique chapter that shows the author's grasp of distributed systems. It encapsulates a lot of hard-won technical experience, which is useful to the developer of distributed systems. This is a rare and valuable commodity. Guidelines on transaction design, business logic design, databases, testing and application design are provided in concise discussions.

Chapter 9 sums up all the concepts in the book by walking the reader through a complex example. The "time tracker" example is well picked. It uses the major features from the EJB spec in a real-world system. It demonstrates the use of both session and entity beans. There is also a session bean (TimeTrackerBean) that in turn uses an entity bean (EmployeeBean). The design is well explained. This chapter is useful since it's hard to find an example that illustrates the different aspects of EJBs. Most tutorial classes fail to provide such examples. This chapter alone makes the book worth buying. Chapter 10 briefly talks about existing EJB vendors and future directions for EJB. Of course, you can compare the predictions with what has really happened since the book is a few months old. An Appendix carries complete code listings. These are also available in the accompanying CD.

In conclusion, this is a good book for programmers and developers who are trying to get an introduction to EJBs and like playing around with plenty of code.

ajit@sys-con.com