1999 Readers' Choice Awards

JAVA DEVELOPER'S JOURNAL READERS' CHOICE AWARD

JAVA™ DEVELOPER'S JOURNAL

The World's Leading Java Resource

Volume: 4 Issue: 8, August 1999

JAVADEVELOPERSJOURNAL.COM

COMPLETE CODE LISTING INCLUDED!

SYS-CON PUBLICATIONS

JAVA DEVELOPER'S JOURNAL

Editor-in-Chief:	Sean Rhody
Art Director:	Jim Morgan
Executive Editor:	M'lou Pinkham
Production Editor:	Cheryl Van Sise
Assistant 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

Jeremy Allaire, Gene Callahan, Brian Clark, Jim Crafton, Scott Davison, Jim Mangione, Gopalan Suresh Raj, Sean Rhody, Lawrence Rodrigues, E Ming Tan, Vivek Sharma, Jason Westra, Alan Williamson

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
Project Coordinator:	Jaclyn Redmond
Advertising Assistant:	Christine Russell
Graphic Designers:	Robin Groves Alex Botero
Graphic Design Intern:	Aarathi Venkataraman
SYS-CON Radio Editor:	Chad Sitler
Webmaster:	Robert Diamond
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. Application to mail at Periodicals Postage rates is pending 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

SEAN RHODY, EDITOR-IN-CHIEF

(Im)perfect Timing

All right, I'm ready to admit that I made a slight miscalculation. Not an error, necessarily…just a slight misjudgment when it came to the timing of something. Back in January I made a set of predictions concerning the industry, as I'm wont to do at the beginning of a new year. In those predictions I stated that I didn't think we'd see any Enterprise JavaBean products until the end of this year.

Unless you've been living under a rock, you know that EJB is the all-Java equivalent of CORBA or COM. It's also the direction I see Java programming headed toward for a number of reasons. First of all, it offers a strong set of built-in functionality. It handles transaction management for you. It handles concurrency management for you. It allows you to write code as if only a single user was ever going to use it. And it provides a fairly scalable platform for distributed computing. Some EJB implementations will run on any hardware that supports a JVM, from a PC running 95 or Linux to an IBM 390.

It was my suspicion that applications and products designed for EJB wouldn't be available until the end of the year. That was for a number of good reasons. The 1.0 release of the EJB specification was publicly released in March 1998. To the best of my knowledge, the first EJB servers appeared around September – please don't get mad if I missed some early releases. It doesn't really matter

I think the interest in EJB started to reach critical mass about the time we did our ***JDJ*** Editor's Choice Awards, which we presented at the JiBE show in December. By that point I had identified a number of vendors who were releasing products that supported EJB, either as the sole point of the server or as an addition to existing functionality. Last time I counted there were 28 server products that supported EJB.

My thoughts in January were that it would take about nine months to develop application products for EJB. I wasn't wrong in that – in fact, I was the chief architect of a project that did just that. What I forgot about was that some vendors were hard at work with beta software three months before the true release of shipping products.

So that brings us to August, or actually July as I write this. I now know of one shipping product and one beta product that are completely based on EJB. The shipping product is called JumpStart, a product from the Theory Center. I'll be evaluating the product in a forthcoming issue. The beta product is from a company called TradingDynamics. Both address e-commerce. JumpStart is a general-purpose transaction mechanism that supports rapid development of business to consumer sites. The TradingDynamics product is more of a brokerage product – designed to handle negotiations and auctions rather than just purchases.

A fact I find particularly interesting is that TradingDynamics and the Theory Center both selected WebLogic as their default EJB host. I have nothing but good things to say about BEA and their WebLogic product – I worked with it for nine months and found it to be incredibly stable and scalable. The Cloudscape Java database also seems to be evolving as a de facto standard for demonstration implementations. For those of you who don't know Cloudscape, they focus on embedded databases for Java. Small ones, not databases that would compete with Oracle and the like. If you have a system that needs a local database with no management, Cloudscape is worth a look.

So I was a little off on timing. I'm glad to see it, actually. There's not a technology related to Java that I believe in more strongly than EJB. I think it's a good answer to distributed computing, and most solutions associated with it provide multiple connection options so you can build your business logic, then create a Java application to administer the system, and an applet or a servlet to provide access to the system.

sean@sys-con.com

Apology: *On the cover of the June isssue writer Mohan Rajagopalan's name was misspelled. We apologize for the error.*

AUTHOR BIO

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

Jeremy Allaire, Vice President of Technology Strategy at Allaire

Java, XML and Web Syndication

Over the past year significant momentum has grown behind the unique intersection of two core Web platform technologies, Java and XML. Clearly, with Java emerging as the predominant Internet-system programming language and XML emerging as the dominant model for Internet data, these two technologies are bound to intersect in interesting ways. Indeed, almost without exception Java has been the primary reference implementation environment for emerging XML technologies such as XML and XSL parsers, as well as XML-centric object database systems.

The meshing of these two key technologies is also at the intersection of a revolution in how the Internet economy functions. Over the last few months we've begun to see an emerging concept that is quickly spreading across leading Internet companies and technology platforms. The concept is Web syndication, which refers to building affiliate and syndicate networks across Web sites, and syndicating content and application assets to create Internet value chains and Internet-based e-commerce businesses.

The convergence of Java, XML and Web syndication is perhaps one of the most important shifts in the Web platform landscape, and beckons to be better understood. Indeed, this shift affects how Java systems will be built for the Web. It affects the future of distributed computing models, potentially shifting the balance of power away from Java-centric models based on RMI, CORBA or EJB and toward XML-centric distributed computing. Overall it creates a larger economic opportunity for every company embracing the Internet.

A quick review of the current state of Java, XML and syndication on the Web is well worth our time.

The Current Role of Java

Despite the significant limitations of Java in browsers, Java has established itself in a dominant position in the emerging Web application landscape. Increasingly, companies are embracing the three (or *n*-tier) application model with HTML/DHTML in browsers, Java on the server and databases behind this. There are many variations on this model, of course, depending on the requirements and available skill sets on any given project. For example, in many cases the server-side includes a scripting abstraction layer, typically implemented in JavaScript or ColdFusion, and tag-based development, which encapsulates servlets or EJB component interactions.

For the most part these applications are built for end users accessing the applications through Web browsers on intranets and Internet sites. Additionally, companies are increasing their investment in Java as a component implementation language through CORBA and EJB, where application objects service other applications across the network. For many developers and companies there is even a sense of market stability in the historically incoherent distributed-computing landscape. Or so it seems.

The Current Role of XML

As an emerging technology, XML has received nearly the same amount of coverage that Java did in its early years of adoption. Despite this, XML adoption remains fairly limited and scattered, though this is rapidly changing as companies begin to understand its role and economic value more clearly.

XML emerged over a year ago as a next-generation technology for structuring and exchanging information on the Web. Initially, XML was greeted as the "future of HTML," with the focus on browser-based adoption and integration with presentation and formatting technologies such as CSS and XSL. This was the document-centric XML world speaking, and, frankly, was easy for pundits and the press to understand. This approach to XML has largely fallen flat; few Web developers are actively using XML for storing Web documents or CSS/XSL to format and deliver those Web documents.

(continued on page 79)

jeremy@allaire.com

Author Bio

Jeremy Allaire is a cofounder and vice president of technology strategy at Allaire. He helps determine the company's future product direction and is responsible for establishing key strategic partnerships within the Internet industry. Jeremy has been a regular author and analyst on Internet technologies for the past seven years, and he holds degrees in both political science and philosophy from Macalester College.

The Java-CORBA Way to Large-Scale Software Design

COMPLETE CODE LISTING INCLUDED!

How to extend the service broker

WRITTEN BY E MING TAN

In the CORBA-based, service broker framework, the data that's required and shared among various heterogeneous systems is coordinated in a synchronized manner by a server process, yet maintained locally by each participating system.

Although lots of articles mention how to design a large-scale system, they often lack implementation details, due partly to the complexity of the issues involved. In this article I'll provide an example of a simple, generic and yet useful implementation of a large-scale system based on Java-CORBA architecture.

Background

Designing and writing large software is very different from writing a program that can be completed in one day. This article will be particularly useful to those of you involved in large-scale software development. My earlier article (*JDJ* Vol. 4, issue 1) proposed a way to design a loosely coupled, highly maintainable software product centered around an object known as the service broker. Thus some terms used in that article will be used here without redefinition or explanation. This article discusses how to extend the service broker based on CORBA architecture and written purely in Java.

CORBA-Based Service Broker

Overview

In a real business situation, supporting computer systems may exist in a heterogeneous environment. It's likely that each system consists of different subsystems, each maintaining their own data using methods chosen by their management. For example, a typical company may have an order entry system, a service delivery system and a billing system. Assuming each system was designed and maintained by different departments, you'll have a big headache if there's no coordination between them. Problems arise when the systems need to share common data. This is when a CORBA-enabled service broker is useful.

Technically speaking, each local application will instantiate its own copy of a client service broker (SBroker), which acts as a local proxy object for the remote object of the service broker. The SBroker proxy object can also become a factory object for each application, if there is such a need. This would reduce the overhead for naming service lookups from the CORBA client application.

The remote service broker server is implemented as a normal CORBA server. A client service broker can communicate via IIOP with a component that's a reachable CORBA object, called a service broker moniter, hosted by the service broker server. If you look at the SBroker code in Listing 1, you'll notice the getMonitor method and mon object of type monitor. This is the method that returns a CORBA object reference of the remote service broker server object, which is mon in this case. Referring to Listing 2, the service broker monitor basically allows a client to:

- Register (add) itself to listening to a specific broadcast event.
- Remove itself from event listening.
- Broadcast an event to (notify) the listening application.

The Implementation

In this design I've chosen a pure Java-based ORB, JacORB (version 0.9f2) courtesy of Gerald Brose (brose@inf.fu-berlin.de) since it's a free open source with more features than the Java IDL of Java 2 (JDK 1.2) from Sun Microsystems. It allows you to test the example in this article without a commercial ORB product. However, the design should work with other commercial ORBs as well with minimal modification.

JacORB is used in the CORBA-enabled service-broker framework in the following areas:

- The COSS naming service to allow distributed listeners to locate the service broker in a standard CORBA way
- To provide the ORB client and the ORB server with listeners to register event types they're interested in receiving and for the event notifications of sources to listeners via the service broker
- Calling back of the service broker's monitor to the listeners – thus the ORB clients (in this case the listeners) have to be CORBA objects too

The service broker server architecture depicted in Figure 1 consists of a monitor CORBA object that has an event manager running as a separate thread, an optional Object Design, an ObjectStore ODBMS database for persistency and an ORB server, which is JacORB in this case, running a naming service as well as an Interface Repository (IR) daemon.

Because broadcast events in this environment are asynchronous, I've used ObjectSpace's JGL 3.0 class library to maintain the events; it's popular and has many standard Java generic classes, plus I'm using JDK 1.1.6 to compile the framework. By the time this article is published, the JGL class library should be part of Java 2 and known as the Collection API.

JGL's Queue class is used in the following manner (see Listing 2):

- To keep event objects in the queue so they're processed in the order they were received by the service broker
- To avoid lost events when more than one source (subsystems) tries to broadcast at the same time
- To delete an event from the queue if the call back inside a listener is returned successfully

To allow a more robust operation in this service broker framework, the server, source and listeners have to operate in a way that facilitates fault tolerance. For instance, whenever the service broker server wakes up (initial start or restarted from the last crash/hanged), it'll always process the event currently on the top of the queue, if any event exists. When a listener wakes up (initial start or restarted from the last crash/hanged), it'll always look for the service broker's monitor object and register itself with the monitor. Whenever a source wakes up (initial start or restarted from the last crash/hanged), on the other hand, it will broadcast a new event via the service broker only if the previous event is processed and removed from the queue by the event manager.

The next JGL class used is HashMap. It's used to keep the object reference of the listeners to facilitate the event-manager thread to call back the listeners. In this design example (see Listing 2) the listeners are associated with their unique application ID as the key.

Alternatively, one can try out Object Manager JGL version 3.1 for ObjectStore, available free from Object Design's Web site. The modified version of the JGL library, which directly supports persistency via a scalable ObjectStore database, is known as dJGL. According to Object Design, ObjectStore PSE Pro 3.0 can recover from an application failure or system crash. If an application (in this case the service broker server) fails during a transaction, when you restart it, it'll return to the way the database was prior to the transaction. If an application fails during a transaction commit, when you restart it, the database will be either the same as it was before the transaction or it will reflect all of the transaction's changes. It depends on how far along in the commit process the application was when it terminated. ObjectStore ODBMS is a full-blown enterprise object database. It's also compatible with JDK1.2 Collection API, but the details aren't covered in this article.

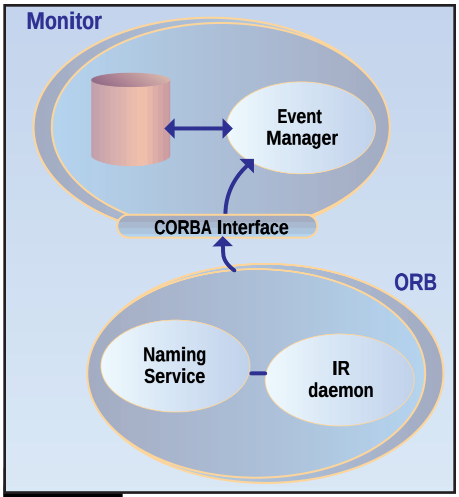


FIGURE 1: A simplified CORBA-based service broker diagram

The Assumptions

Before I show you the codes and explain them in depth, I'll share the two most important assumptions I've made in this design. While these assumptions may not be true in your application environment, they serve the purpose here.

- A source can never broadcast a new event until the previous one has been processed by the event manager and consumed by the listening application.
- Applications with the same application ID can't listen to a single event; in other words, each application can only listen to a particular event that's closely associated with its application ID.

If you need to broadcast more than one event at any time, a slight modification of the service broker server codes is needed. A more involved change is necessary if you wish to broadcast a specific event to more than one listening application. Some may argue that the word *broadcast* itself is a misnomer, since broadcast often means the target is more than one entity. Remember, the example given here is an oversimplified implementation. If there's a need to broadcast to more than one listening application using the same event, the event manager has to call back all the applications that are interested in the event before removing the event from the queue. Multiple sources broadcasting to a listener with the same event is also not advisable, though it's possible with some modifications.

A Typical Scenario

The easiest way to show you how to code under this CORBA service broker framework is to imagine how it works in a typical business environment. Imagine there are a few systems in this environment and they all communicate with the service broker server via IIOP protocol. All applications generally keep their own data, but they don't have to. Figures 2 and 3 illustrate the steps involved in this process.

When the service broker starts up, it will bootstrap its CORBA object and make it available to all source and listener applications (see Figure 2a). In this case the server registers its monitor object with the COSS naming service. After the server is ready, the framework is ready for virtually any source application to broadcast an event as well as any listening application to process it.

Assume that Application 2 needed to send data to Application 1. Typically, Application 1 will have to register itself with the service broker server. Remember, in this framework every application will have a local service broker proxy called SBroker. That's why you see a D-shaped capsule-like structure on top of each application. SBroker encapsulates the CORBA implementations from all participating applications in this framework. In other words, a source application that needs to broadcast an event (publish an event via the service broker server) to a listening application has to instantiate an SBroker object. The listening application has to do the same thing in order to subscribe to the event (pushed by the server once it's available). This is done by registering itself as a listener with the server (see Figure 2b).

The following code snippet shows you how to instantiate the monitor object from an application and obtain the remote object reference:

```
monitor mon = (new SBroker()).getMonitor();
```

Once the remote instance of a service broker object is obtained, registering listeners or broadcasting an event are straightforward steps. To register as a listener (see Listing 3 for a sample listerner code), an application invokes addRadioBroadcastListener of the monitor object. The following code snippet shows you how to do that:

```
//=== instantiate a copy of the listening application
RbListener rbListener = new RbListener();
//=== register itself as one of the radio listeners
boolean ret = rbListener.mon.addRadioBroadcastListener(rbListener);
---------------------- (1)
```

If you need to remove an application from listening, the application invokes the removeRadioBroadcastListener method of the monitor object as follows:

```
//=== remove itself from listening if there is a need
rbListener.mon.removeRadioBroadcastListener(rbListener);
```

How does a source broadcast an event? By notifying the service broker (see Figure 2c), which is done by invoking the notifyRadioBroadcastListeners method of the monitor object. The following code snippets show you that:

```
//=== instantiate a copy of the source application
RbSource rbSource = new RbSource();
//=== assuming that event id is equal to application id
RbEvent evt1 = new RbEvent("800");
---------------------- (2)
//=== setting the event type for listening application
evt1.setEventType(rbSource.getEventID1());
---------------------- (3)
//=== broadcast now
boolean ret = mon.notifyRadioBroadcastListeners(evt1);
---------------------- (4)
```

As you've noticed by now, to broadcast an event via the service broker server by sending a notification message involves more steps than becoming a listener. In line 2, we're basically creating an event object to be sent to the listener. In this case it's assumed the event ID is the application ID (which is 800). An application ID is just a unique number assigned manually by the application developer. If the same ID has already been allocated to another registered application, registration of the new listener with the

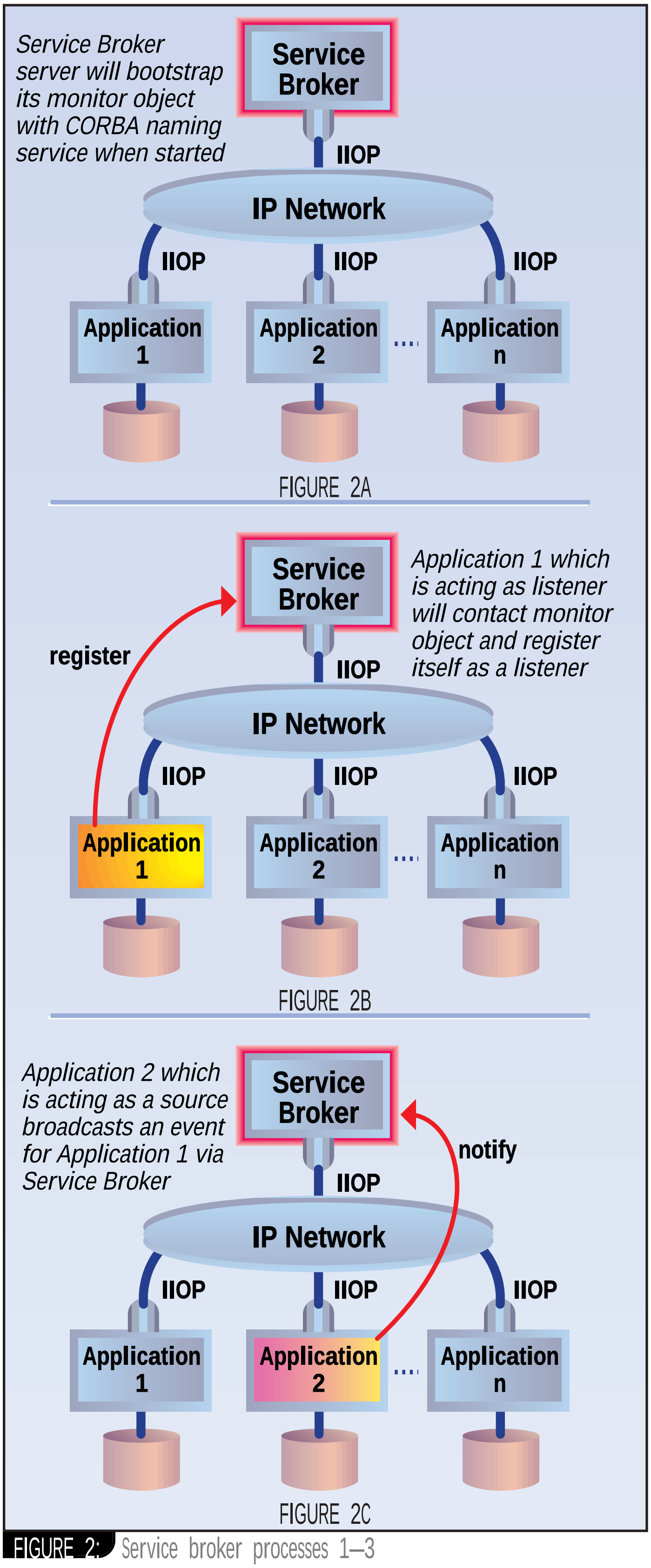

Service Broker server will bootstrap its monitor object with CORBA naming service when started
Service Broker
IIOP
IP Network
IIOP
IIOP
IIOP
Application 1
Application 2
....
Application n
FIGURE 2A
Service Broker
Application 1 which is acting as listener will contact monitor object and register itself as a listener
register
IIOP
IP Network
IIOP
IIOP
IIOP
Application 1
Application 2
....
Application n
FIGURE 2B
Application 2 which is acting as a source broadcasts an event for Application 1 via Service Broker
Service Broker
notify
IIOP
IP Network
IIOP
IIOP
IIOP
Application 1
Application 2
....
Application n
FIGURE 2C

FIGURE 2: Service broker processes 1–3

monitor will fail (see line 1). Next (in line 3), we set the type of the event, since an application can have multiple data that needs to be processed in different ways. Each event type may be associated with one or more data. The real pushing of an event to the service broker is done in line 4. Nevertheless, if the previous event (which has an ID of 800) is still not processed by the listener, this method call would fail with a return status of false.

The role of the service broker server is to receive events from the source and push them to the listener by calling back the broadcastPerformed method of the listener. In our scenario the service broker will call back Application 1 (shown in Figure 3a). Thus a listener must be a type of org.omg.CORBA.CORBject object too (see Listing 4).

What follows is the most critical step as far as the business environment is concerned – the exchange of data between applications. Application 1 (listener application) retrieves the data from Application 2 (source application) via a peer-to-peer communication (see Figure 3b).

Surprisingly, this framework makes no assumption of any method to achieve that. You may ask why. A simple answer is because it's meant to be generic! Nevertheless, one may find getDataRetrieveMethod and getDataStoreMethod methods of the RbEvent class useful (see Listing 5). They're the placeholders where you can specify how to retrieve the data from the source application and/or how to store the data in the listening application (e.g., if the data format is different). It may include properties like a host name, FTP server name, data server name, user name and password, or even jdbc strings. It can even be the data itself, e.g., the XML data. It's up to you to decide which methods you're most comfortable with or are most suitable for your specific environment.

The following code snippet shows you an example of the inside of a listening application:

```
public String getDataRetrieveMethod() {
    return
    "sourceFileDir=/WebApp/Download"
    + "&"
    "sourceFile=T9806812.txt"
    + "&"
    "ftpUrl=146.135.30.167"
    + "&"
    "ftpUsr=A04"
    + "&"
    "ftpPwd=A04A04";
}
public String getDataStoreMethod() {
    return
    "listenerFileDir=/usr/local/WebApp/Download"
    + "&"
    "listenerFile=T9806812.txt";
}
```

In the example above, the source application happened to be a legacy system with no other means of communication except FTP protocol. You'll notice that the getDataRetrieveMethod returns enough information for retrieval purposes. In this case the listening application is a UNIX box and the downloaded file is local as far as the listener is concerned. Therefore it doesn't need to include much information in the getDataStoreMethod method. Of course, a proper way to set all these properties is via setter methods such as setDataRetrieveMethod and setDataStoreMethod.

Last, as soon as the data is fully transferred from Application 2 to Application 1, the callback returns control to the event manager of the service broker. When this happens, the event manager removes the event object from the queue and continues with the next event, if there's any in the queue (see Figure 3c). The whole process repeats itself relentlessly (from Figures 2b to 3c), and now we have a new service broker framework based on CORBA.

What Did I Miss?

Now, after going through one specific scenario, I guess you have lots of questions. It sounds too easy and is it of any practical value in a business environment? Yes and no. The answer is Yes if you're looking for a generic framework and you don't mind further customization. If you're looking for a full-blown framework, you'll be disappointed. You may want to consider some other commercial products available. This framework doesn't specify how data is exchanged among various sources and listeners. Furthermore, since it's meant to be simple, each event type is forced to directly associate with the listener's application ID. The listen-

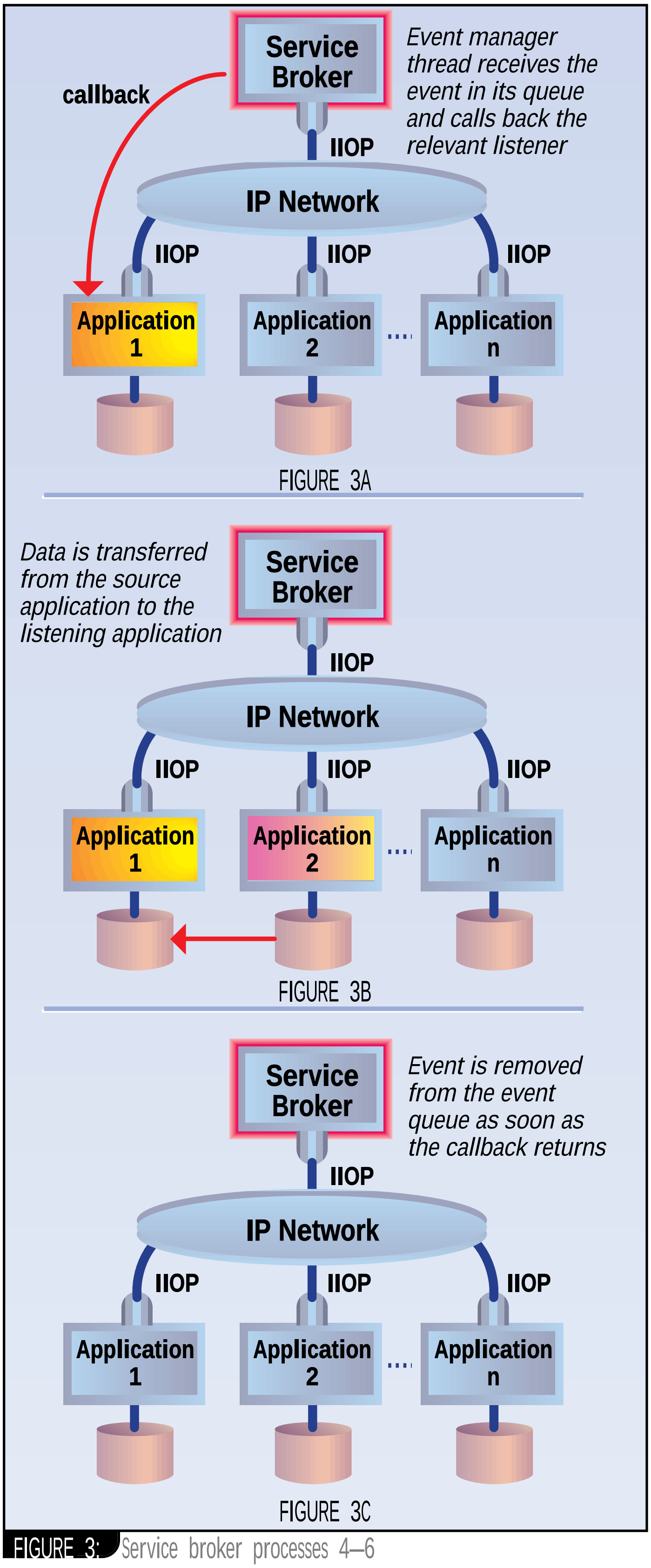


FIGURE 3A

FIGURE 3B

FIGURE 3C

FIGURE 3: Service broker processes 4–6

er's registration with the service broker server would fail if another listener with the same ID has already registered. New-event generation from a source application is blocked until the previous event is consumed by the listener application. In other words, the event buffer size is only size 1 and the event is considered consumed as soon as the callback of a corresponding listener (broadcastPerformed method) returns.

The event manager is very simple too. Currently, a listener's object reference isn't purged from the HashMap even if the listener crashed or died, resulting in an error when the listener restarted and tried to register itself again with the same application ID. If this happens, the easiest way to work around it is to restart the service broker server and all applications involved in the framework. Due to a similar problem, if a source tries to broadcast an event to an invalid listener (crashed/died or not running for whatever reason), the listener-bound event isn't processed and other subsequent events (if any) are blocked. The event should have been discarded if its associated application ID is no longer valid and the source application should be informed about the status of the event. Another way of overcoming this is to set a certain timeout period for each event processed so that if the corresponding listening application is unreachable, the event will be rendered invalid. Once the event is made invalid, the source application will realize this just before it's about to generate a new event for the listener.

Other Considerations

One of the most critical features of an enterprise solution is the transaction. The processing of events by listeners and the subsequent removal of the events from the queue is not an atomic operation. What happens if after a listener has processed an event it fails just before the event is removed from the queue by the event-manager thread? How does it affect the integrity of the data when the listener wakes up again? To answer these questions, I used ObjectStore PSE Pro and the persistent version of JGL class library, dJGL, which enabled me to provide a scalable transactional capability to this service broker framework.

The following code snippet shows how minimal additions (in blue) into the service broker server codes (see Listing 2) will provide the transactional support commonly required in business environments:

```
...
//=== use djgl class library instead of jgl
import com.objectspace.djgl.*;
//=== add ObjectStore package too
import COM.odi.*;
public class monitorImpl extends Thread implements monitor
    ...
    Database db = null;
    Transaction tx = null;
    monitorImpl() {
        Session.create(null, null).join();
          db = create("Queue.odb");
          tx = Transaction.begin(ObjectStore.UPDATE);
         queue = new Queue();
          ...
    }
    private static Database create(String dbName) {
          try {
                  Database.open(dbName,
ObjectStore.OPEN_UPDATE).destroy();
          } catch (DatabaseNotFoundException e) {
          }
          return Database.create(dbName, 0777);
    }
    ...
    public void run() {
          ...
          while( true ) {
                tx = Transaction.begin(ObjectStore.UPDATE);
                  queue = (Queue)db.getRoot("queue");
                  if ( !queue.isEmpty() ) {
                            ...
                  }
                  tx.commit();
           }
     }
...
```

Besides the transactions, you should consider the security issues as well. A mechanism needs to be in place so that only valid or authorized sources or listeners can broadcast events or register and receive the events, respectively. Other enhancements may include cross-system exceptions mapping and a centralized logging facility.

Conclusion

In a nutshell, this CORBA-based service broker framework allows required data that's shared among various heterogeneous subsystem applications to be synchronized by a server process, yet maintained locally in each application. It provides centralized control of the means of retrieving and storing data advertised to the service broker by the participating applications. It also provides a standardized way for a CORBA-based interface as an interprocess communication among various enterprise applications. Best of all, it allows companies not only to keep their existing "legacy" systems but to extend their functionalities as well as their roles, via Java and distributed CORBA-based technology.

Reference

Lewis, G., Barber, S. and Siegel, E. (1998). *Programming with Java IDL.* Wiley Computer Publishing.

Author Bio

E Ming Tan was the chief architect of corporate intranet-based paging applications at Singapore Telecom. He has worked with Cable & Wireless, Inc., and is currently working with MCI Worldcom on a Web-based customer billing system. He can be reached at futurewave@iname.com.

futurewave@iname.com

Listing 1

```
(see source code file SBroker.java for details):
public class SBroker {
   public monitor getMonitor() {
   }
   monitor init() {
   }
   public static void main( String[] args ) {
   }
   static void out(String msg) {
   }
}
```

Listing 2

```
(see source code file monitorImpl.java for details):
public class monitorImpl extends Thread implements monitor {
    monitorImpl() {
    }
    public sbroker.listener getApp(int id) {
    }
    public int getAppID(sbroker.listener rbListener) {
    }
    boolean getEventProcessed(sbroker.event evt) {
    }
    public boolean
notifyRadioBroadcastListeners(sbroker.event rbEvent) {
    }
    public boolean addRadioBroadcastListener(sbroker.listener
rbListener) {
    }
    public boolean removeRadioBroadcastListener(sbroker.lis-
tener rbListener) {
    }
    private void out(String msg) {
    }
    public void run() {
    }
}
```

Listing 3

```
(see source code file RbListener.java for details):
class RbListener extends Thread implements RadioBroadcastLis-
tener {
     public RbListener() {
     }
     public int getAppID() {
     }
     public void broadcastPerformed(sbroker.event rbEvent) {
     }
     public static void main(String argv[]) {
     }
     public void run() {
          while(true) {};
     }
}
```

Listing 4

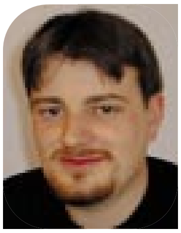
```
(see source code file RadioBroadcastListener.java and sbro-
ker.listener.java for details):
public interface listener extends org.omg.CORBA.CORBject {
    void broadcastPerformed(sbroker.event rbEvent);
}
```

Listing 5

```
(see source code file RbEvent.java for details):
public class RbEvent extends java.util.EventObject implements
java.io.Serializable, sbroker.event {
    public RbEvent(String eventID) {
    }
    public String getEventID() {
    }
    public void setEventType(String evt) {
    }
    public String getEventType() {
        return eventType;
    }
    public String getDataRetrieveMethod() {
    }
    public String getDataStoreMethod() {
    }
}
```

Party! Party! Party!

An eclectic assortment of Java hors-d'oeuvres



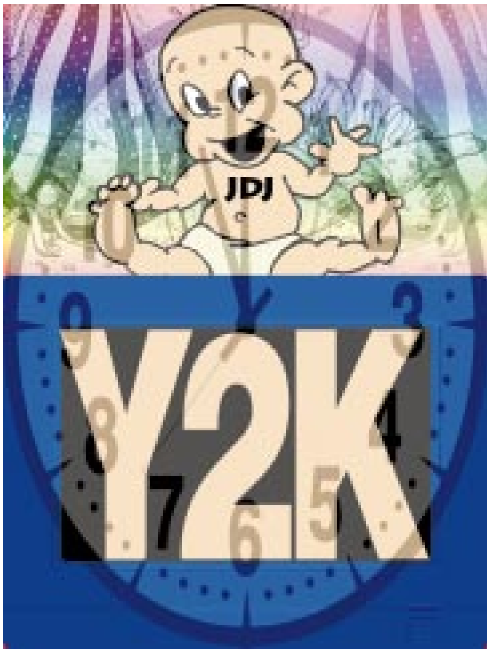
WRITTEN BY
Alan Williamson

don't know about you, but these months are shooting by at a tremendous rate of knots. Here we are again, into the latter half of the year…and I was just getting used to being back after Christmas. It's all very exciting, racing up to the day that dare not speak its name: yes, the big millennium turnover. Boy, am I looking forward to that day!

But on to more important matters of state: churning out this month's column. This is the month after the biggest month in our developers' calendar: JavaOne. If I am a good boy and manage to finish the show report for this issue, you'll find it lurking somewhere in these pages not so far away from this column. That said, I shall not bore you twice and make you read the same drivel over again.

As my regular readers will recall, we're always on the lookout for developers who can do a day's work without our having to tell them constantly what Java really is. Well, this month we have a new recruit starting with us, and so far so good. He's performing a sterling job. The frightening thing is that he has a personality, which doesn't fit your typical hardcore programmer. I guess we'll have to beat that out of him!

Anyway, why am I telling you this? Before sending our new soldier to the frontline to battle with client code, we put him on some internal projects that needed attention. Nothing too taxing. Basically he had to create a couple of additional classes based on some established core classes. He performed admirably and well within the time allocated. I asked him how the testing went. He said wonderfully well, that all the methods worked. Fantastic.



Hold on…what do you mean all the methods worked? On closer inspection it transpired that our new man had merely called the top-level methods and never checked for anything going awry internally. Bless him. Such blind faith. Now this wasn't his fault directly. He had never really tested code before, and since his test case worked it didn't occur to him to test a scenario that might be a little out with the necessary parameters.

But who can really blame him? It reminded me so much of me when I started on my long journey to Java fulfillment. I was scared to test my code in case my beautiful creation didn't live up to my expectations. I just assumed – hoped is more like it – that it would stand its ground in all environments. Or was it more laziness? Hmmm…the more I think about it, the more I better leave this thread alone before my client base loses complete faith in me!

That Left-Out Feeling

I know. I promised I wouldn't mention JavaOne, but please excuse me. I have to get this out of my system.

Being located in one of the far-flung corners of the globe, Scotland, you sometimes feel left out. You know, the one that gets picked last for the school football team, or the one that never gets invited to the party. You know there's a whole world going on out there, but you're just never sure that you'll fit in. Well, running the n-ary empire from here, we sometimes feel the party is getting down somewhere else and the world has forgotten all about Java and has moved on without telling us. So when an opportunity like JavaOne raises its head, I make the annual pilgrimage to San Francisco.

It's not for personal gratification, you understand. It's for the good of the company (I just want to get that point across in case any of my directors are reading this and trying to figure out why I have absconded to California for a jolly).

There's really only one reason to attend JavaOne. Ironically, it's not to hear people talk, or to walk around booths packed full of vendors peddling their wares. No. It's to meet people or – as we say in the corporate jungle – "to network."

There's a rich tapestry of individuals that make up this wonderful Java universe we've chosen to partake of, and I had the privilege of meeting some of them. I'll introduce some of the more colorful ones to you here.

For those of you that attended and wondered which one I was, that's an easy one to answer. Think hard. Do you remember up in the Media Mall there were lots of big, stupid-looking costumes walking around, depicting scenes of Java (Java in the loosest sense of the word, let me assure you)? Well, I wasn't one of them. However, I was the one with the Scottish kilt loitering around the SYS-CON Radio panel. You may have been too scared to come up and introduce yourself – and who'd blame you? But for those that were brave enough, I thank you. It was indeed a pleasure to be able to put a face to a lot of you.

One of the first people I had the good fortune to meet and get to know rather well over the course of the week was one Rick Ross. Rick is a man I've shared column space with, being another ***JDJ*** columnist himself, but I had never met the man in the flesh. What a character Rick is! After meeting the man, I made a bet with myself that should he ever run for office, not much would stop him. He's a born talker, which is one of the reasons the Java Lobby is so popular. For those of you not aware of the Java

Lobby, get yourselves to the Web site, www.javalobby.org/, and join the crusade. I see Rick as the thinking man's Jimmy Hoffa, rallying around us mere mortals, making sure Java is heading on a clear and set course. If you ever get the chance to meet him, pin the bugger down and ask him anything that concerns you about Java. I'll bet my grandmother's left leg that Rick has a thought on it. Go on, try it. And if he seems annoyed, you never read this. We never had this conversation!

But having a blether with Rick was a joy. He has many of the same thoughts that we outpost developers hold. Which assured me that maybe the party wasn't going on without us.

Back to My Pilgrimage

One of the things about developing so remotely is that, unlike our Silicon Valley counterparts, the chance of us literally bumping into the competition is highly unlikely. (Unless, of course, Dolly, our amazing cloned sheep, has had a Java chip implanted in there by those clever scientists at Edinburgh.) That aside, while standing at the SYS-CON booth waiting for something that at this point escapes me, a man sidled up and started to talk to me. I think he was complimenting me on my column, but I could be mistaken. Anyway, the conversation drew to a close and for some reason I asked for his business card. When I read it I was bowled over. It was probably the initial shock at seeing his contact information that has made the first point of contact so blurry. (You know who you are, so if I have this slightly wrong, then please forgive me.)

The card I held before me represented our biggest competitor. First time this ever happened to me. Of course, once I introduced myself and the particular project we competed with them on, he knew who we were. We then went for a sit-down while he fetched his CEO. It was all quite amicable. We danced around one another like male peacocks trying to prise information out of one another. So that's what the enemy looks like....I jest.

I have to give a big thanks to Mary Hancock and Clint Dalton for allowing me to hang with them over a number of evenings. Being European, I felt like a fish out of water at times in the throes of California, so these budding reporters from ServletCentral.com kept me on the straight and narrow. I'd like to give them full credit for trying. Mary had her camera with her all week and was clicking lots of showpieces. Every so often Mary and Clint attempted to get conclusive pictures of what really lies under a Scotsman's kilt. Well, I can proudly say the legend of what is underneath there remains a mystery.

So meeting people at JavaOne was the most important thing. Be proud of the industry you're in, and make sure you get yourself out there. There are some excellent characters about, and I could fill a whole magazine just chattering about them, but I won't. This would spoil the fun for you. But before I leave this, if you're ever around a Java Lobby meeting or up near Seattle, be sure to look up what have to be the coolest brothers in the world of Java: the Ramadan lads from 4thpass. Mazin and Zeyad are the Java equivalent of Bill and Ted, and I highly recommend you speak to these guys if you ever get the chance.

> "I think the brainwashing at Oracle has gone a little too far and way too much information has been erased."

Mailing List

At JavaOne many of my readers and subsequent mailing-list attendees came up and said hello. They assured me they'd be inspiring the list with many new and varied topics of conversation. With that in mind, here's my monthly plug for the list. To join, send e-mail to listserv@listserv.n-ary.com with subscribe straight_talking-l in the body of the message. From there you'll get instructions on how to participate.

Salute of the Month

There are many salutes I could award this month and some of them I mentioned earlier in this column. But one person I have to thank is Jim Driscoll, from the ranks of management at Sun. Now let me explain this. I met Jim in person over a year ago, and had been in constant e-mail contact with him for around eight months earlier than that. Jim is one of the original Java Servlet architects, and it is through this that we had gotten to know one another. For me, Jim was the embodiment of what we perceive a Silicon Valley geek to look like: long hair, unshaven, knee-deep in code and always going somewhere to check his e-mail. Well, I nearly didn't recognize him when he came up to me at JavaOne. He's gone all corporate. The hair is cut, the face is shaved and he's no longer coding. He's management now, climbing that corporate ladder. Jim's professional coding days are gone and he's now resigned to watching as others wrestle with the joys of threads and other goodies Java has hidden up her sleeve. So Jim, we salute you for a job well done. May the rung above you always be free!

Book Review

Last month I promised you a review of Larry Ellison's book once I finished it. Well, I finished it, and what a read it turned out to be! I can't even begin to tell you the sort of antics our man at Oracle has been up to.

Regular readers will know the problems I (and many of you) experienced with Oracle's JDBC drivers. Now I know why. If you read this book be prepared for an amazing tale. I was keen to learn if the contents of the book were true so I went off to the Oracle booth at JavaOne to get some insider information. I have to say I am rather impressed by the way none of the Oracle people I spoke to had any idea the book even existed, let alone the contents. They acted dumb very well. Or maybe they weren't acting, because the same people had never heard of the magazine you're holding in your hands now – the biggest Java circulation magazine in the world and they had never heard of it. Hmmm, not convincing, methinks. I think the brainwashing at Oracle has gone a little too far and way too much information has been erased. But I did manage to speak to some ex-Oracle employees, including a couple that actually were mentioned in the book, and they assured me it's true. So all I can say is read it, and get back to me on our mailing list. I'd love to talk about it.

Now that I'm safe and sound back in my corner here in the lowlands, I can start being paranoid again about the party that no one is inviting us to. Keeping my eye on these supercloned Java sheep, I bid you farewell. Catch you next month.

alan@sys-con.com

AUTHOR BIO

Alan Williamson is CEO of n-ary (consulting) Ltd, the first pure Java company in the United Kingdom. A Java consultancy company with offices in Scotland, England and Australia, they specialize solely in Java at the server side. Alan is the author of two Java Servlet books and contributed to the Servlet API. He can be reached at alan@sys-con.com (www.n-ary.com) and welcomes all suggestions and comments.

JDJ FEATURE

COMPLETE CODE LISTING INCLUDED!

JavaBeans vs Enterprise

WRITTEN BY LAWRENCE RODRIGUES AND GOPALAN SURESH RAJ

JavaBeans has been at the center of many new paradigms and technologies that have emerged since its inception. Among emerging technologies, Enterprise JavaBeans has generated tremendous interest in the business computing community. However, a common misconception is that an Enterprise JavaBean is an extension of a "plain vanilla" JavaBean with enterprise functionality.

While both JavaBeans and Enterprise JavaBeans are software component models, their purpose is different. A JavaBean is a general-purpose component model, whereas EJB, as the name suggests, is a component model that is enterprise specific. Even though these models have entirely different architectures, they adhere to certain underlying principles that generally govern a software component model. We'll use these principles and the basic characteristics of software components to compare JavaBeans and Enterprise JavaBeans.

Goals

The underlying theme of both JavaBeans and Enterprise JavaBeans is, as Sun puts it, "Write once, run anywhere" (WORA). Accordingly, the primary objective of both models is to ensure portability, reusability and interoperability of Java software components.

JB

JavaBeans takes a low-level approach to developing reusable software components that can be used for building different types of Java applications (applets, stand-alone apps, etc.) in any area. Whether you're developing a simple applet or a complicated application, JavaBeans can be integrated into your system with ease. Already a large number of vendors offer JavaBeans in a variety of fields. Some JavaBeans may be common to multiple fields. A chart bean, for instance, can be used in scientific, engineering and business computing applications.

EJB

Enterprise JavaBeans takes a high-level approach to building distributed systems. It frees the application developer to concentrate on programming only the business logic while removing the need to write all the "plumbing" code that's required in any enterprise application. For example, the enterprise developer no longer needs to write code that handles transactional behavior, security, connection pooling, networking or threading. The architecture delegates this task to the server vendor.

What Are Beans?

JB

A JavaBean – a bean – is a reusable software component that can be visually manipulated in a builder tool. Builder tools help you assemble applications by visually connecting beans. This doesn't mean you can't build applications in the conventional way. If you prefer, you can hand-code applications by using beans. When you build applications by visual connection, no coding is involved (though in semivisual tools you may have to write some code).

Builder tools also help you customize beans visually. Once customized, you can save beans as serialized prototypes (.ser files). When beans are part of an application, they run just like any other object. They are instantiated differently, though, as saved beans have to be resurrected from their serialized prototypes.

EJB

An Enterprise JavaBean – an EJB – is a reusable server-side software component. Enterprise JavaBeans facilitate the development of distributed Java applications, providing an object-oriented transactional environment for building distributed, multitier enterprise components. An EJB is a remote object, which needs the services of an EJB container in which to execute.

Architecture

JB

The JavaBean specs define a component model to build, customize, assemble and deploy general-purpose Java software components. In the JavaBean model, the structure and behavior of a bean is described by three basic features: properties, methods and events.

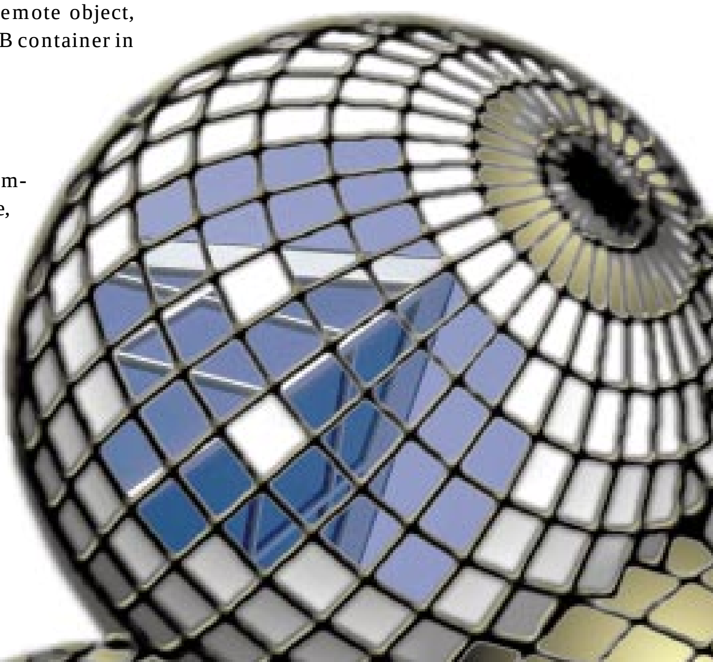
Properties are a bean's named attributes that can be edited to customize a bean. *Methods* describe a bean's behavior. *Events* serve two purposes:

1. ***Bean connection:*** When a bean is running in a builder tool, events enable visual connection.
2. ***Notification:*** When a bean is running in an application, events notify occurrences and pass data from source to target.

When a bean is inserted in a visual builder tool, its exposed properties, methods and events are discovered through the twofold process called *introspection*, which involves:

1. Discovery from the explicit information, which is provided by the bean provider through a bean-specific class called BeanInfo.
2. Automatic discovery of a bean's features by using the reflection API. To facilitate this, methods in the bean have to adhere to certain naming conventions as stated in the JavaBeans specs.

The JavaBeans 1.0 specification dealt mainly with the design-time behavior of JavaBeans. It was later augmented by a set of three specifications code-named "Glasgow" to address some runtime issues. The "Extensible Runtime Containment and Services Protocol" spec, which is part of Glasgow, describes the relationship between a bean and its container at runtime. As shown in Figure 1, a bean that conforms to this spec can be part of a nested containment structure and can utilize arbitrary services from its container.



JavaBeans

The primary objective of both models is to ensure portability, reusability and interoperability of Java software components

EJB

The Enterprise JavaBeans spec defines a server component model and specifies how to create server-side, scalable, transactional, multiuser and secure enterprise-level components. Most important, EJBs can be deployed on top of existing transaction processing systems including traditional transaction processing monitors, Web, database and application servers.

As shown in Figure 2, a typical EJB architecture consists of:

- An EJB server
- EJB containers that run on these servers
- Home objects, remote EJB objects and Enterprise beans that run in these containers
- EJB clients
- Other auxiliary systems like the Java Naming and Directory Interface (JNDI), the Java Transaction Service (JTS) and Security services

Unlike JavaBeans that use introspection, the EJB container uses the EJBMetaData class to query an EJB for its metadata at any given time.

Some of the advantages of pursuing an EJB solution are:

- EJB gives developers architectural independence.
- EJB is WORA for server-side components.
- EJB establishes roles for application development.
- EJB takes care of transaction management.
- EJB provides distributed transaction support.
- EJB helps create portable and scalable solutions.
- EJB integrates seamlessly with CORBA.
- EJB provides for vendor-specific enhancements.

Application Development Roles

JB

Even though the JavaBeans specification doesn't mention application development roles explicitly, we can infer the following two roles:

1. ***Bean provider***: develops, customizes and packages beans
2. ***Bean user***: customizes, assembles and deploys beans

EJB

The EJB specification assigns specific roles for project participants charged with enterprise application development utilizing EJBs. For instance, business developers can focus on writing code that implements business logic. Deployers of EJB can take care of installation issues in a simple and portable fashion. The server vendor can take care of providing support for complex system services and make available an organized framework for an EJB to execute in, without assistance from EJB developers. The EJB specification defines six primary roles:

1. ***Enterprise Bean provider***: develops the enterprise bean
2. ***Application assembler***: connects the beans together and packages them
3. ***Deployer***: deploys the packaged bean on the server
4. ***EJB server provider***: provides a framework that can execute the EJB containers with transactional support
5. ***EJB container provider***: provides tools to generate the container classes that encapsulate the bean at runtime
6. ***System administrator***: responsible for configuration and administration of the environment on which the EJB server executes

APIs

Both the JavaBean and the EJB specs define APIs for bean development, execution and deployment.

JB

- ***java.beans:*** The APIs from the original JavaBeans specification are implemented in the java.beans package, which includes classes and interfaces needed for both bean providers and visual builder tools.
- ***java.beans.beancontext:*** The APIs from "The Extensible Runtime Containment and Services Protocol" spec are implemented in the java.beans.beancontext package, which includes classes and interfaces for implementing bean context services, bean contexts and bean context children. This package is available only in Java 2.

Both are core Java packages and are therefore available in different vendor implementations of Java.

EJB

- ***javax.ejb:*** The APIs defined in the Enterprise JavaBeans specs are included in the javax.ejb package, which is a Java standard extension. In addition to this, EJB relies on the APIs defined for the Java Transaction API (JTA), the Java Transaction Service (JTS) and the Java Naming and Directory Interface (JNDI).

Bean Characteristics

Structure and Behavior

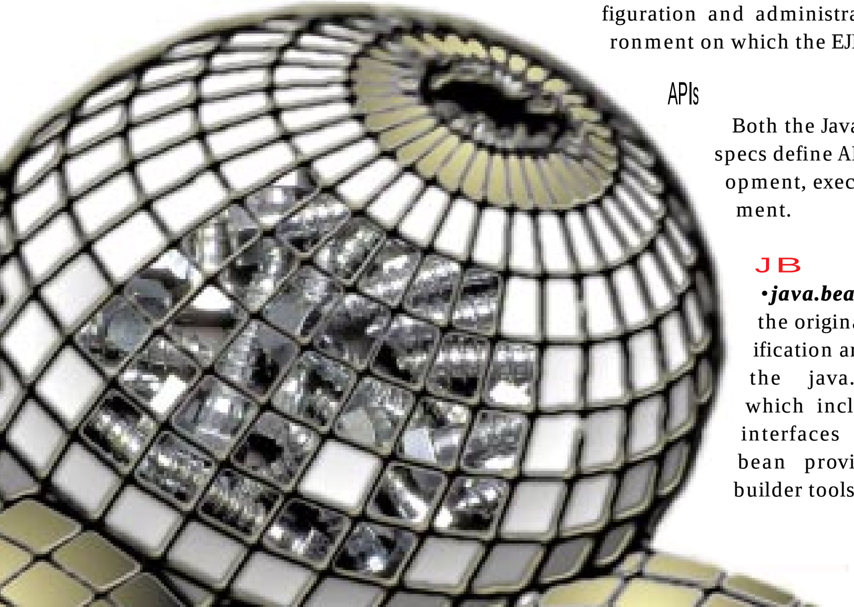
JB

A bean, which is identified by a class, can encapsulate any functionality. While the bean class need not extend any other class, it needs to implement the serializable or externalizable interface either directly or through inheritance. Even though a single class identifies a bean, its functionality can be spread over many classes through inheritance and delegation. An important restriction on the bean class is that it should have a no-argument constructor.

Since a bean's internal structure and implementation details aren't exposed, you can't extend a bean's functionality as is. You can, however, customize it to suit your application. To extend a bean's functionality, you need to create a new bean by extending the existing bean class. In other words, you need to use the bean as a class library.

As we alluded to in the Architecture section, a bean's interface to the outside world (which includes visual builder tools, other beans and applications) is through its properties, methods and events. This interface is at the bytecode level, which means there is no need to recompile the bean when it's assembled in an application.

For the properties, methods and events to be discovered through introspection, the related methods have to follow certain signatures



and naming conventions specified in the JavaBeans specs. In addition, a bean provider can also furnish feature descriptors through a design-time-only class called BeanInfo, which is specific to a bean class.

EJB

An EJB typically encapsulates business logic that operates on data. An EJB's interface to the outside world is through its Home and Remote interfaces. While the Home interface defines a factory to create new beans and find existing beans, the Remote interface defines the business methods that the bean supports. Each packaged EJB is identified by its Home interface and its Home object, its Remote interface and its EJB object, the enterprise bean class implementation and its deployment descriptors.

In EJB there's no need for a BeanInfo class because the deployment descriptors in conjunction with the EJBMetaData class take care of the bean description.

As mentioned earlier, an EJB isn't represented by a single class, but by the enterprise bean implementation, its home interface and its remote interface.

A typical Home interface for a hypothetical ParentEJBean would look like this:

```
import javax.ejb.*;
import java.rmi.*;

public interface ParentHome extends EJBHome {
Parent create(char relationship, String name)
   throws CreateException, RemoteException;
Parent findByPrimaryKey(ParentPK parentKey)
   throws FinderException, RemoteException;
}
```

You can't inherit the Home interface because of a problem with the create() methods; that is, the child will need to supply the same create() methods as the parent, but the methods will return different values (or remote interfaces). Unfortunately, the Java language doesn't permit a class to have two methods that differ in signature only by return type, so inheriting from Home interfaces is out of the question.

You can, however, inherit implementations. You could have a ParentEJBean implementation class and declare a ChildEJBean as:

```
public class ChildEJBean extends ParentE-
JBean {

}
```

You can thus reuse implementation code. But with this approach you'll run into the differing-only-by-return-type problem in the ejbCreate() methods of bean-managed persistence (see the sections on Types and Persistence). This is because the methods will take the same arguments, but will return different primary keys. Since create() methods in Session beans return a void, they don't encounter this problem. Container-managed EJB's ejbCreate() methods also return a void. Even though they compile, there may be problems when the EJB container generates code that actually returns the primary key.

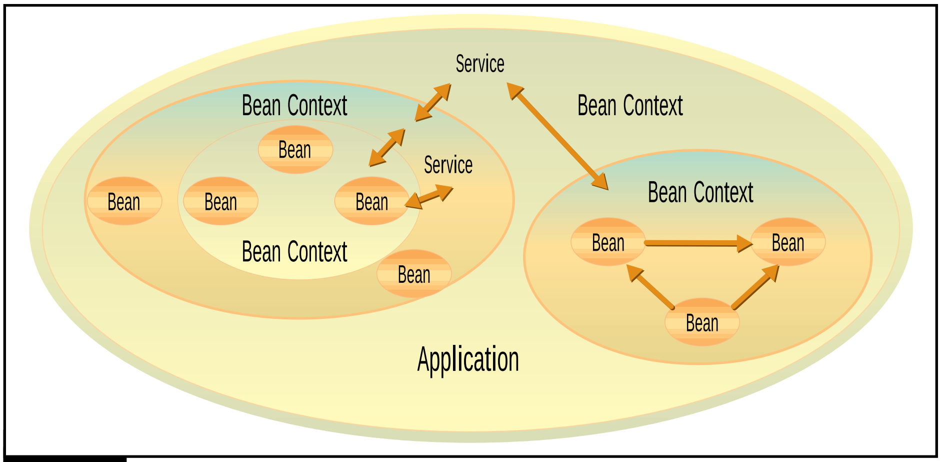


FIGURE 1: A typical Java application with beans and bean contexts

Inheritance in EJB is tricky, and you're better off using containment instead.

Visibility

JB

A bean can be visible, invisible or both. A Stopwatch bean, for instance, can have the GUI shown when it runs on the client side and turned off if it's running on the server side. Even if a bean is invisible, it can be customized, serialized and connected to other beans in visual builder tools.

EJB

An EJB is a nonvisual remote object that resides only on the server side.

Types

JB

Beans that conform to JavaBeans 1.0 specs aren't typed. The Glasgow spec, however, allows two types of beans: Simple and Participant. A Simple bean isn't aware of its container, whereas a Participant bean actively participates in its container. A bean that conforms only to the original JavaBeans 1.0 specs falls under the Simple bean category. A Participant bean, however, conforms to "Extensible Runtime Containment and Services Protocol" specs in addition to the original bean specs. A Participant bean can also discover and utilize arbitrary services from its container.

Table 1 gives a comparison of Simple and Participant beans.

EJB

There are two primary types of EJBs: Session and Entity beans. While an Entity bean has a unique identity defined by its primary key class, a Session bean has no unique identity. Multiple clients can thus share an Entity bean. A Session bean, on the other hand, is created, used and destroyed by the client that created it. Each bean has associated with it a context object (Session context or Entity context) for its lifetime. Table 2 compares Session and Entity beans.

There are two types of Session beans: stateful and stateless. Similarly, there are two types of Entity beans: container-managed persistent entities and bean-managed persistent entities. (See the following section for more details.)

Persistence

JB

Beans are persistent. In the context of JavaBeans, beans are objects that can be saved and resurrected. Persistence is achieved by saving a bean's internal states through serialization. As mentioned before, a resurrected serialized prototype of a bean can be included in an application.

EJB

Stateful Session beans may have internal states. Therefore, they need to handle activation and passivation. Passivation is the process by which the state of a bean is serialized out into secondary storage. Activation is the process by which it is deserialized from secondary storage. These types of EJBs can be serialized and restored across client sessions. To serialize, a call to the bean's getHandle() method returns a handle object. To restore, a call to the handle object's getEJBObject() method is used to return a bean reference.

Entity beans are inherently persistent beans. There are two types of persistence in Entity Beans:

- ***Bean-managed persistence:*** In BMP the Entity bean is directly responsible for saving its own state. The container doesn't need to generate any database calls. Hence, the programmer needs to hard-code persistence into the bean through explicit JDBC or embedded SQL calls.
- ***Container-managed persistence:*** In CMP the EJB container is responsible for saving the bean's state. Since it's container managed, the implementation is independent of the data source. The container-managed fields need to be specified in the deployment descriptor and the EJB container automatically handles persistence.

Customization

JB

Beans are visually customizable. You can customize a bean by editing its properties. Visual builder tools typically present property sheets for this purpose. To generate property sheets, builder tools use introspection. The JavaBeans spec also provides an alternative to property sheets. It specifies an interface called Customizer to enable bean providers to build bean-specific customizers. Such a customizer can also be invoked at runtime. (See L. Rodrigues's article, "On JavaBeans Customization," ***JDJ*** Vol. 4, issue 5, for more details.)

EJB

EJB customization is a bit different from JavaBean customization. There's no concept of a property sheet or a custom-written customizer for an EJB. EJBs are customized using deployment descriptors, which define the contract between the ejb-jar provider and the EJB consumer. It captures the declarative information (information not included directly in the EJB code) that's intended for the consumer of the ejb-jar file.

The two types of information in the deployment descriptors are the EJB's structural information and application assembly information. In EJB 1.1 XML is used to define the deployment descriptors. EJB vendors may provide tools that can be used by the ejb-jar provider to create deployment descriptors.

Containment and Nesting

JB

A bean can contain another bean. The original JavaBeans 1.0 specs didn't explicitly address containment-related issues. The Glasgow specification defines the notion of a logical bean container or BeanContext (see Figure 1). A child bean in a container can itself be a BeanContext, thus allowing nesting of beans. In a BeanContext child beans (Simple and Participant) can be dynamically added and removed. They can also access arbitrary services from the container.

EJB

EJBs always run within an EJB container. EJBs request different services from their containers and are aware of their environment. Containers can't contain other containers and therefore there's no concept of nesting in EJBs. Each EJB is associated with a context object (either a SessionContext or an EntityContext that provides information about the EJB). The context object is the component's handle on the container, through which the component can get transaction information, security information and information from the component's deployment descriptor. The EJB component calls into the Context object through the SessionContext or EntityContext interface.

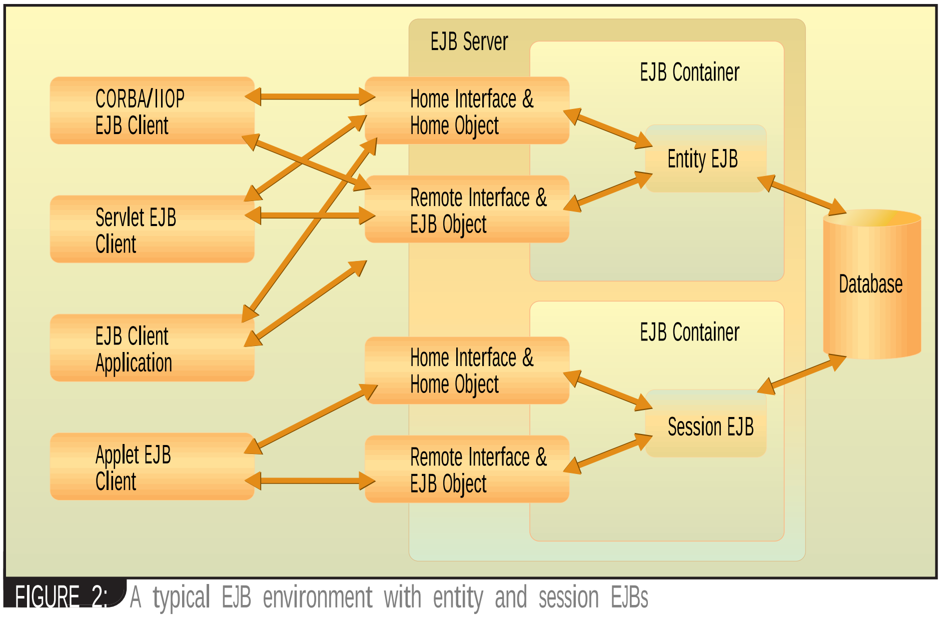


FIGURE 2: A typical EJB environment with entity and session EJBs

Packaging and Deployment

JB

Beans are packaged in JAR files. The bean provider has to provide a manifest file with JavaBeans-related attributes in order to identify the bean class. A JAR file can hold more than one bean. However, the JAR entry for each bean class should have the Java-Bean attribute set to true. An example:

```
Name: Spreadsheet.class
Java-Bean: True
```

Two more JavaBean-related attributes are Depends-On and Design-Time-Only. A typical bean JAR file contains design-time and runtime bean classes, documentation, resources such as images, and sound files.

EJB

EJBs are also packaged in JAR files. To identify the EJB class, the bean provider has to provide a manifest file in which the jar-entry for the EJB class should have the Enterprise-JavaBean attribute set to true. An example:

```
Name: ~gopalan/BankAccountDeployment.ser
Enterprise-Bean: True
```

Application Assembly

JB

You can compose applications by visually connecting beans in a builder tool or manually by writing connection programs. The application so developed can be an applet or a standalone application. Beans for an application need not come from the same vendor because beans can be developed independent of one another.

As we alluded to before, events act as interfaces between beans. Bean connections are performed at design time and are unidirectional. The source bean fires an event and the target bean receives it. When the application assembler chooses a source bean for connection, the builder tool discovers through introspection the events fired by that bean. When the application assembler chooses a target bean for the selected source bean, the builder tool discovers the compatible methods in the target bean, again through introspection.

EJB

EJBs are assembled into larger deployable applications. The input of application assembly is one or more ejb-jar files produced by different providers. The output is one or more ejb-jar files that contain Enterprise beans with their assembly instructions. As we mentioned in the Customization section, the application assembly instructions have been inserted into the deployment descriptors. EJBs too can be developed independent of one another. Once an EJB's home and remote interfaces are known, you can use them to create or find them and to invoke methods on them.

Execution

JB

The execution phase consists of the instantiation and running of beans.

Instantiation

Even though a bean is an object, it is instantiated differently. Instead of the new operation, beans are instantiated using the Beans.intantiate() method. There are many flavors of this method. A bean instantiation example:

```
// Obtain the Class Loader
ClassLoader loader =(Account.class).get-
ClassLoader ();

// Instantiate the Account Bean
Account account = (Account)Beans.instanti-
ate (loader, "Account")
```

Running

As mentioned earlier, a bean is like any other object when it's running in an application. Normally, methods in a bean aren't directly invoked from other beans. The bean connections determine what methods need to be invoked. Event adapters enable indirect method invocation.

Beans can also be executed in a builder tool at design time. As mentioned before, an important requirement for a bean to run in a builder tool is for the bean class to have a constructor with no arguments. This is because the builder tool can't provide the constructor parameters. When there are many constructors, it can't decide which one to use.

Java 2 has new Beans.instantiate() methods to facilitate the instantiation of applets and BeanContexts.

EJB

The EJB execution phase consists of locating, instantiating and running.

Locating the EJB

EJB clients locate the specific EJB container that contains the enterprise Bean through the JNDI. They then make use of the EJB container to invoke bean methods. As you may be aware, JNDI allows multiple directory services to coexist and even cooperate within the same JNDI client. Using JNDI, a user can navigate across several directory and naming services while seeming to work with only one logical federated naming service.

Instantiating the EJBean

Once EJB clients obtain a reference to the Home object, they can create the EJB by calling its create() method or find EJBs by calling its find methods. This creates the EJBObject and the EJB component inside the EJB container.

Invoking Methods on the EJBean

The EJB client can now use the remote object reference to invoke methods on the EJBean by invoking its remote methods, which form the business logic of the component. For example:

```
// get the JNDI naming context
Context initialCtx = new InitialContext ();

// use the context to lookup the EJB Home
interface
AccountHome
home=(AccountHome)initialCtx.lookup
("com/gopalan/Account");

// use the Home Interface to create a Ses-
sion Bean object
Account account = home.create (1234,
"Athul", 1000671.54d);

// invoke business methods
account.credit (1000001.55d);
```

Characteristics	Simple Bean	Participant Bean
Nesting	Not aware of its container, so nesting structure isn't exposed.	Aware of its container. Nesting structure is exposed through BeanContext APIs.
Types	Not typed	BeanContext, BeanContextChild, BeanContextService
Access to arbitrary services	No	Yes
InfoBus Aware	No	Yes

Table 1: Comparison of Simple and Participant beans

Transactions

JB

There is no explicit transactional support.

EJB

- ***Declarative transaction management:*** The EJB container vendor is required to provide transaction control. The EJB developer who is writing the business functionality needn't worry about starting and terminating transactions. However, for maximum flexibility, the EJB spec provides for declarative transaction management. Six declarative modes can be specified by the deployer: TX_NOT_SUPPORTED, TX_BEAN_MANAGED, TX_REQUIRED, TX_SUPPORTS, TX_REQUIRES_NEW, TX_MANDATORY.
- ***Distributed transactional support:*** EJB provides transparency for distributed transactions. This means that a client can begin a transaction and then invoke methods on EJBs present within two different servers running on different machines, platforms or JVMs. Methods in one EJB can call methods in the other EJB with the assurance that they'll execute in the same transaction context.

Security Services

JB

There are no special security APIs for JavaBeans.

EJB

EJB provides authorization using the Java security model. EJB server implementations may choose to use connection-based authentication in which the client program establishes a connection to the EJB server. The client's identity is attached to the connection at connection establishment time. The EJB/CORBA mapping specifies that the CORBA principal propagation mechanism be used. This means that the client ORB adds the client's principal to each client request. The communication mechanism between the client and the server propagates the client's identity to the server. Security in EJB 1.1 is declaratively defined in the deployment descriptors and is role based.

Characteristics	Session	Entity
Client access	Single	Multiple
Identity	Not unique	Unique defined by entity's primary key
Survives container crash?	No	Yes
Transaction-aware	May or may not	Yes
Database Access	May or may not	Yes
Lifetime	Limited to that of the client	Alive as long as data exists in the domain model

Table 2: Comparison of Session and Entity beans

Interoperability

JB

JavaBeans can interact with components built using other models, which includes the widely used Component Object Model (COM). Using the Beans-ActiveX Bridge, a bean can be converted to an ActiveX control. A converted ActiveX control can interoperate with other ActiveX controls in an ActiveX container.

EJB

While the EJB spec allows the EJB server vendors to use any communication protocol between the client and server, the EJB/CORBA mapping document is prescriptive with respect to what goes on the wire. This allows both system-level and application-level interoperability between products from vendors who choose to implement the EJB/CORBA protocol as the underlying communication protocol.

Java clients will optionally communicate with server components using IIOP. They'll have a choice of APIs – either the Java RMI or the Java mapping of the CORBA IDL interface. Non-Java clients communicate with server components using IIOP and the appropriate language mapping. Clients wishing to use the COM+ protocol communicate with the server component through a COM-CORBA bridge. Also realize that the client of an EJB can itself be a server component (for instance, Java Server Pages or a servlet), so an HTTP-only Web client can use a servlet to make EJB invocations.

An EJB can't be deployed as an ActiveX control because those controls are intended to run at the desktop and EJBs are server-side components. CORBA-IIOP compatibility via the EJB-to-CORBA mapping is defined by the OMG. However, EJB components may be able to communicate with DCOM servers using a DCOM-CORBA bridge.

Summary

Table 3 summarizes the characteristics of JavaBeans and Enterprise JavaBeans.

Characteristics	JavaBeans	Enterprise JavaBeans
Purpose	General purpose	Enterprise computing
Description	A reusable software component capable of being manipulated in visual builder tools.	A reusable server-side component.
Structure and Behavior	Represented by a single class, but functionality can be spread over multiple classes through inheritance and delegation. Bean class should have no-arg constructor. Can be a local or remote object.	A remote object described by a Home interface and a Home object, a Remote interface and an EJB object, an enterprise bean implementation and its deployment descriptors. Primary keys are used to identify Entity beans.
Visibility	Visible, invisible or both	Invisible
Types	No types in JavaBeans 1.0 Glasgow has Simple and Participant beans.	Session and Entity beans
Persistence	A bean class has to be Serializable or Externalizable	Session beans can be activated and passivated. Persistence in Entity beans is of two types: bean managed and container managed.
Customization	A bean is customized by setting its properties.	An EJB is customized by setting deployment descriptors
Containment	Simple beans aren't aware of the container whereas Participant beans are. Participant beans can be nested and can request arbitrary services from the container.	EJBs run within an EJB container and have complete knowledge of the environment. EJBs can't be nested. EJBs can request arbitrary services from their containers.
Application Assembly	Can be assembled manually or visually.	No tools currently support this, but vendors may come out with tools that do very soon.
Packaging	Packaged in JAR file. Bean class attribute: Java-Bean	Also packaged in JAR file. Bean class attribute: enterprise-javabean
Security	No special security. Follows the Java security model.	Security is declaratively defined in the deployment descriptors and is role based
Transaction Support	No explicit support	May be transaction enabled – six modes TX_NOT_SUPPORTED, TX_BEAN_MANAGED, TX_REQUIRED, TX_SUPPORTS, TX_REQUIRES_NEW, TX_MANDATORY
Interoperability	Bean can be converted to an ActiveX control through the ActiveX bridge and can in turn communicate with other ActiveX controls.	CORBA-IIOP compatibility via the EJB to CORBA mapping defined by the OMG. COM interoperability through COM-CORBA bridges.
APIs	java.Beans and java.Beans.BeanContext	javax.ejb

TABLE 3: Summary of differences between JavaBeans and Enterprise JavaBeans

Conclusion

JavaBeans technology, now in its third year, has undergone the acid test of the industry. The original spec has gone through a couple of iterations since its introduction. Although the spec may undergo further iterations, many aspects of JavaBeans have been firmed up.

On the other hand, the EJB spec is just more than a year old and still evolving. Now at version 1.1, it provides an excellent architectural foundation for building distributed enterprise-level business object systems. Some areas in the spec need to be examined closely, however – most notably in the EJB model for handling persistent objects. Standardizing the contract between development tools and systems to provide a uniform debugging interface for all development environments is being considered as well. The specification will still go though more iterations before becoming final.

The other issue is compatibility. There are two areas where compatibility is an issue. One is what actually constitutes an "EJB-compatible" server. The other is guaranteeing that EJBs developed on servers from different vendors can interoperate.

References

1. Cable, L. (1997). Glasgow Specification Version 99A. Sun Microsystems, July.
2. Enterprise JavaBeans Specification 1.1. Sun Microsystems, June 1999.
3. Hamilton, G., ed. (1997). JavaBeans Specification, Version 1.02. Sun Microsystems, July.
4. Rodrigues, L. (1997). "Java, The Next Generation: JavaBeans." *Java Developer's Journal*, Vol. 2, issue 1, January.
5. —(1998). *The Awesome Power of JavaBeans*. Manning.
6. Seshadri, G., and Raj, G.S. (1999). *Enterprise Java Computing – Applications and Architecture*. SIGS/Cambridge University Press.

Author Bios

Lawrence Rodrigues, author of The Awesome Power of Java Beans, *has more than 15 years of industry experience. You can reach him through his "Bean man's home page" at www.execpc.com/~larryhr or by e-mail at larryhr@execpc.com.*

Gopalan Suresh Raj, a senior analyst at Compuware Corporation, is a contributing author to Enterprise Java Computing – Applications and Architecture *and* The Awesome Power of JavaBeans. *His expertise spans enterprise component architectures and distributed object computing. He can be reached at www.execpc.com/~gopalan or by e-mail at gopalan@execpc.com.*

larryhr@execpc.com gopalan@execpc.com

Enterprise JavaBeans **Persistence**

Bean-managed and container-managed persistence: advantages and disadvantages

COMPLETE CODE LISTING INCLUDED!

WRITTEN BY
JASON WESTRA

As I sit down to write this article, the hype for *Star Wars Episode 1: The Phantom Menace* is even greater than the Java industry hype before the release of the Enterprise JavaBeans specification 1.0! I couldn't help including a few references of my own to the beloved trilogy in hopes of adding a little flavor to an otherwise drab topic: persisting data.

Enterprise JavaBeans Persistence: Two Persistence Models

"Would you like the window or the aisle, sir?" asks the Intergalactic Travel ticket agent. Decisions, decisions! Does it matter? Well, maybe. That depends on a number of factors. Hmmm....Are you flying into the magnificent city of Coruscant or bound for barren Tatooine? Is the seat adjacent to the window seat occupied by Darth Maul?

Life is all about options, and the development of business applications is no different. Fewer options amount to less flexibility; however, limiting your choices certainly simplifies the decision process. On the other hand, more options means more flexibility for your application, albeit at the cost of lengthy debate over which choice is best for you. This month's topic discusses the choices available when you implement persistence with entity beans in your Enterprise JavaBeans application.

Entity Bean State Management

The Enterprise JavaBeans 1.0 (and newly released 1.1) specification attempts to simplify decisions about persistence by providing a straightforward persistence model for managing the state of entity beans (note that entity bean support is not mandatory in 1.0 of the EJB specification, but is mandatory for EJB 1.1 compliance). State management refers to the creation, destruction, loading and storage of persistent data in an entity bean. The EJB specification defines a small set of methods to perform these important services: ejbCreate, ejbLoad, ejbStore, ejbRemove, ejbActivate and ejbPassivate. We're particularly interested in the first four, which represent the CRUD (Create, Read, Update and Delete) operations typically seen in a persistent application. Over the course of this article you'll become intimate with these methods and their state management responsibilities through the Intergalactic Travel ticket system examples.

Besides describing the methods for managing an entity bean's state, the EJB specification offers two models of persistence: bean managed and container managed. However, as we've noted, multiple options can lead to confusion and extra decisions you have to make to meet your project's deadline...which was yesterday. Each model has its advantages and disadvantages, of course. This article highlights the semantics and trade-offs of the two EJB persistence models, providing you with ammunition to decide quickly and easily which one best suits your needs.

The Intergalactic Travel ticket system provides a good illustration of the two persistence models. For our purposes, a simple entity bean that represents a persistent luxury liner ticket will suffice. Figure 1 reveals the UML class diagram of the TicketEntityBean; Table 1 depicts the RDBMS storage schema for this entity bean. This month's example enterprise beans were written to the EJB 1.0 spec and deployed using a BEA WebLogic Application Server 3.1.4.

Bean-Managed Persistence

Bean-managed persistence refers to situations in which an entity bean manages its own state through the implementation of persistence methods via JDBC, object serialization, etc. To code an entity bean with bean-managed persistence, you must implement the relevant operations to create, store and load your entity bean from its persistent storage format. Let's take a look at an example entity bean from the Intergalactic Travel ticket system that implements bean-managed persistence. The jdj.Ticketing.beanmanaged.Ticket-EntityBean (TicketEntityBean for short) contains JDBC SQL calls to handle storage and retrieval of the bean from our MS Access RDBMS.

Intergalactic Travel ticket agents issue new tickets as passengers reserve seats and pay in credits for the flight. The life cycle of our TicketEntityBean begins when we invoke a create method through our factory interface class, TicketEntityHome. An entity bean's home interface can provide multiple "create" methods, each having a corresponding ejbCreate() method on its entity bean. This method is called from the home interface class via its container, and allows you to initialize your entity bean accordingly. In our example we have just one create method, which takes all attributes necessary to create a fully formed TicketEntityBean. Listing 1 demonstrates how the ejbCreate() method for our bean-managed TicketEntityBean not only sets the entity bean's attributes, but also establishes a connection to the database, prepares a JDBC statement and executes an insert statement into the persistent store. The TicketEntityBean contains helper methods to promote reuse of code. They're shown in Listing 2.

Next, to access our existing TicketEntityBean, we go through the TicketEntityHome once again and call the home.findByPrimaryKey(TicketEntityPK pk) method. In the case of our bean-managed persistence entity bean, this call is forwarded to the ejbFindByPrimaryKey() method of an empty instance of the TicketEntityBean. TicketEntityBean.ejbFindByPrimaryKey() reads in the appropriate information from the ticket table in MS Access and populates the entity bean. Eventually we receive a remote reference, TicketEntity, from the TicketEntityHome. Just as in the ejbCreate() method, you have to write all the code necessary to access the database.

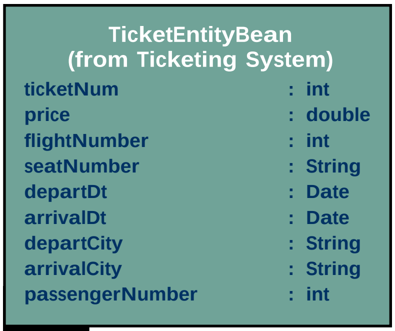
JDBC calls to fetch our entity bean data are located in a single helper method, read(). The entity bean may be loaded in two ways: ejbFindByPrimaryKey() and ejbLoad(). Following good OO form, the code is written in a single read() method and called from each "load" method to achieve localization and reuse. Listing 3 demonstrates calling read() from our ejbFindByPrimaryKey() method.

Now that we have our TicketEntity reference, we can modify it by changing the arrival city (e.g., from Coruscant to the Mos Eisley Spaceport on Tatooine) or the flight number. After an operation is performed and its transaction committed, the TicketEntityBean's container will issue an ejbStore() to save the current state of the entity bean to persistent storage. Invoking ejbStore() on our TicketEntityBean will cause the following actions: get a connection to the database, prepare an update statement and execute a SQL Update through JDBC to the ticket table. Again, you have to code all of the relevant logic by hand for SQL database access, connection management and exception handling. Listing 4 demonstrates a bean-managed update.

What if a ticket is invalid and an Intergalactic Travel ticket agent wishes to delete the bad ticket from the database? In EJB you have the flexibility to remove an entity bean in two different ways: indirectly, through the entity bean's Home Interface.remove(myPrimaryKey pk) method, or directly, through the entity Bean.remove() method. Either way, the correct entity bean is located by the container and the ejbRemove() method is called. In the case of a bean-managed entity bean, you must code the JDBC calls to effectively delete the row from the ticket table. This code is illustrated in Listing 5.

Finally, to allow a ticket agent to see a list of valid tickets issued for a particular space flight, we need to provide a mechanism to query our persistent store of tickets and return this list. To do so will require a search method against the ticket table by flight number.

In EJB, finder methods on entity beans provide the means to return an enumeration of valid results. Finder methods are located on the home interface of the appropriate entity bean; thus, to search for all tickets issued for a specific flight, we have created a method, findTicketsByFlight(int aFlightNumber), on TicketEntityHome, our bean-managed entity's home interface. As usual, the home interface class finder method delegates to an entity bean instance to do the dirty work. The code necessary to return this list is executed in the TicketEntityBean.ejbFindTicketsByFlight() method (see Listing 6). Within ejbFindTicketsByFlight() a JDBC ResultSet is returned from a query, and primary keys for each ResultSet row are instantiated and returned. The home interface class uses this list of primary keys to fetch to the appropriate entity bean instances and return an enumeration of TicketEntity proxies to the caller.



TicketEntityBean (from Ticketing System)	
ticketNum	: int
price	: double
flightNumber	: int
seatNumber	: String
departDt	: Date
arrivalDt	: Date
departCity	: String
arrivalCity	: String
passengerNumber	: int

FIGURE 1: UML class diagram of TicketEntityBean

Bean-managed persistence has numerous admirable qualities including development flexibility and the ability to allow developers to write database access code tailored for performance. For instance, complex searches can be coded into entity bean ejbFinder methods, and entity beans representing data from multiple sources can be coded easily through bean-managed persistence (if your EJB server provider doesn't support mapping multiple tables to a single container-managed entity bean). In addition, developers can access their JDBC ResultSet directly, and tune this code for high performance as needed.

However, it clearly takes considerable work to create a bean-managed entity bean. Besides being tedious to design and implement, managing persistence inside your own beans exposes you to coding errors and limits portability and reusability.

With bean-managed persistence you must code all database access yourself. This can lead to error-prone code and a retesting nightmare when your entity bean's attributes change. You must remember to handle database exceptions correctly and always close resources that might be left hanging from an error.

You must also make your own connection to the database, which may involve creating a new connection each time if your EJB server doesn't provide a mechanism to return existing connections from a pool. Creating a new connection for each database call is costly and ill advised in a high-volume environment.

The topic of ensuring EJB portability would take a whole article or two in itself (in an upcoming issue); essentially, managing your own connections reduces bean portability. Do you see any hard-coded references to WebLogic in our example? What if we decide to port our entity bean to another EJB server? All references to WebLogic would have to be found and replaced.

Finally, what if you wish to port your bean to another storage mechanism? If you'd like to reuse the bean-managed TicketEntityBean in a file-storage schema, you're in for some heavy code modifications!

To solve these problems, EJB provides container-managed persistence.

Container-Managed Persistence

Container-managed persistence relies on the EJB server to generate the appropriate code to manipulate your entity bean. Specifically, the entity bean's persistence services are handled by its container, which transparently manages the bean's persistent state. No data storage coding is demanded from developers of container-managed entity beans. Developers simply write business logic within the entity bean and leave the responsibility for mapping a storage schema in the entity bean's deployment descriptor up to the bean's deployer. It's important to note that the EJB 1.0/1.1 specification doesn't lay the groundwork for how container-managed persistence is to be implemented; thus each EJB server typically provides unique implementations of these services through proprietary containers. While one may store entity beans only as serialized objects, another may offer a whole gamut of options ranging from file-based persistence to storage in an RDBMS or an ODBMS.

So how does a container-managed entity bean differ from our bean-managed entity bean? To understand the differences we'll look at the entity bean class: jdj.Ticketing.containermanaged.TicketEntityBean as well as jdj.Ticketing.containermanaged.file.TicketEntityBean (TicketEntityBean, for short). These classes represent the same bean-managed TicketEntityBean just discussed, but delegate persistence to their containers instead.

Container-Managed TicketEntityBean

A glance at TicketEntityBean.ejbCreate() in Listing 7 reveals the first of many dramatic differences in the bean-managed implementation versus the container-managed implementation of our TicketEntityBean. Initialization of its attributes with the values passed is the extent of the logic performed by the container-managed entity bean. After the execution of ejbCreate(), the entity bean's container will act on behalf of it,

Column Name	Data Type
TicketNum	Number–Primary Key
Price	Currency
FlightNumber	Number
SeatNumber	Text
DepartDt	Date/Time
ArrivalDt	Date/Time
DepartCity	Text
ArrivalCity	Text
PassengerNumber	Number

TABLE 1: Access ticket table schema for Intergalactic Travel ticket system

performing an insert into the appropriate storage device (in our case a file or an MS Access DB). No coding of complex JDBC calls, connection management or exception handling was required to save our container-managed entity.

Similarly, we see that the JDBC code from our bean-managed entity bean has been removed from the following methods: ejbLoad(), ejbStore() and ejbRemove(). In each case the container executes the appropriate code for the retrieval, storage and removal of our TicketEntityBean.

Finally, the helper methods used in the bean-managed example are unnecessary in this case. We no longer need the custom read(), getConnection(), close(), static initializer of our database driver class or the ejbFinder method ejbFindTicketsByFlight(). Functionality previously offered in these methods is handled transparently by the container-managed entity bean's container!

Component-Based Development Is the Key

You'll notice no difference between the JDBC and file-based container-managed TicketEntityBean classes (including primary key, home and remote interface classes [not shown but available for download at www.JavaDevelopersJournal.com]). In fact, a VDIFF on the entity bean files will show no change except for the package name (made for demo purposes only) on all files. Component-based development tools allow us to change the bean's functionality at deployment rather than during development – and all without a single code change.

Enterprise JavaBean's container-managed persistence provides tremendous advantages over bean-managed persistence. Its component-based approach to writing and deploying persistent objects reduces development time by shielding you from writing complex storage code. This allows you to concentrate on business logic within the bean instead of connection management, file resource management, data access and exception handling.

Container-managed persistence also promotes bean portability by not locking your entity beans into a particular storage scheme. Not one line of code had to be modified to deploy our entity bean from an RDBMS storage schema to a file-based schema. Compare that to what needs to be done to convert our bean-managed TicketEntityBean. I'll leave that exercise to you…and may the Force be with you!

However, while container-managed persistence is a powerful option, it does fall short in some areas. For instance, some EJB servers may provide robust mapping of entity beans to multiple tables while others may only allow a one-to-one mapping of table-to-entity bean. In this case you must either write a bean-managed entity bean or write multiple container-managed entity beans and guarantee they always work within the same transaction.

In addition, complex queries may not be possible with the finder method capabilities of today's EJB deployment facilities. The EJB specification 1.0 did not state how to declare container-managed finder methods; thus each EJB server vendor has a different means of deploying your entity bean's finder methods. One vendor may provide functionality that another does not. The EJB 1.1 standard based on XML will hopefully provide more harmonious deployment services between vendors.

Container-Managed Persistence: Today and Tomorrow

The EJB specification is just that, a specification. It allows different vendors to implement the required interfaces as needed. EJB server vendors have been providing container implementations to handle container-managed persistence to RDBMS and file-based stores.

Today, ODBMS vendors such as Versant Corporation (www.versant.com) are assuming the role of EJB Container Provider by developing containers that understand how to save entity beans to their object databases. Currently, third-party containers such as the VERSANT Enterprise Container are being built for specific EJB server vendors such as the BEA WebLogic Application Server. However, once the EJB specification provides guidelines for container implementations, third-party containers will also be portable across EJB servers. These containers will plug into existing EJB servers, providing value-added functionality to your application out-of-the-box.

On the deployment side of entity beans, rumor has it that Symantec's (www.symantec.com) VisualCafé product will soon include modules that offer a common deployment interface to popular EJB servers such as IBM's WebSphere and BEA WebLogic's application servers. Thus you'll be able to deploy a bean to either without needing to understand the nuances of each EJB server's entity bean deployer.

Conclusion

We've covered the two models of persistence offered in the EJB specification 1.0/1.1 – bean managed and container managed – and the advantages and disadvantages of each. Just as a window seat may be optimal on one flight and suboptimal on the next, one model or a mixture of the two will provide a best fit for your application's needs.

Bean-managed persistence offers you the ability to perform complex queries or map entity beans to multiple data sources, yet it lacks portability, reusability and simplicity, requiring knowledge of complex storage mechanisms and connection management. Container-managed persistence has its faults as well, such as a lack of entity bean deployment standards that allows vendors to offer various degrees of functionality in their products. However, the component-based features of container-managed persistence promote entity bean portability and reuse while simplifying the development of persistent objects, allowing you to focus on business logic. The end result: more robust applications built in record time at a fraction of the cost of traditional development techniques!

With EJB you have the flexibility to choose the correct persistent model for your needs. Recognizing that too many options can result in one's downfall, I hope you have been enlightened sufficiently concerning EJB persistence that you'll be able to make the right choice quickly, and soon be off doing fun stuff like coding Java…or viewing *The Phantom Menace* again and again!

AUTHOR BIO

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

jwestra@uswestmail.net.

Listing 1: Bean-managed Create

```
// ejbCreate() inserts Entity Bean
public TicketEntityPK ejbCreate

(int aTicketNum, double aPrice,
 int aFlightNumber, String aSeatNumber,
 Date aDepartDt, Date aArrivalDt,
 String aDepartCity, String aArrivalCity,
 int aPassengerNumber)
throws CreateException
{

// set attributes into Entity Bean to
// insert
    ticketNum = aTicketNum;
    price = aPrice;
    flightNumber = aFlightNumber;
    seatNumber = aSeatNumber;
    departDt = aDepartDt;
    arrivalDt = aArrivalDt;
    departCity = aDepartCity;
    arrivalCity = aArrivalCity;
    passengerNumber = aPassengerNumber;

// mark as modified
    isModified = true;

    Connection con = null;
    PreparedStatement ps = null;
    try{
       con = getConnection();
       ps = con.prepareStatement(
    "insert into Ticket (ticketNum,
    price, "+ "flightNumber, seatNumber,
    "+ "departDt, arrivalDt, departCity,
    "+ "arrivalCity, passengerNumber) "+
    "values (?, ?, ?, ?, ?, ?, ?, ?, ?)");

// fill values from TicketEntityBean
// into PreparedStatement
       ps.setInt(1, ticketNum);
       ps.setDouble(2, price);
       ps.setInt(3,flightNumber);
       ps.setString(4,seatNumber);
       ps.setDate(5, departDt);
       ps.setDate(6,arrivalDt);
       ps.setString(7,departCity);
       ps.setString(8, arrivalCity);
       ps.setInt(9,passengerNumber);

       if (ps.executeUpdate() != 1)
       {
       throw new CreateException
       ("Failed to insert ticket:
                           "+ticketNum);
       }
       TicketEntityPK pk =
       new TicketEntityPK(ticketNum);

        return pk;
    }
    catch (CreateException ex){
       throw ex;
    }
    catch (SQLException sqlEx) {
        throw new CreateException
        (sqlEx.getMessage());
    }
    finally {
       close(ps, con);
    }
}
```

Listing 2: Helper methods for TicketEntityBean

```
// Static initializer in TicketEntityBean
// class
static
{

// initialize weblogic.jdbc.jts.Driver
   new weblogic.jdbc.jts.Driver();
}

// Entity Bean helper methods

private void close
(PreparedStatement aPS, Connection aCon)
{
// final attempt to close the
// PreparedStatement and Connection
  try {
    aPS.close();
    aCon.close();
  }
  catch (Exception ex) {}
}

public Connection getConnection()
  throws SQLException
{
// get an open DBconnection from the
// ejbPool
  return DriverManager.getConnection
    ("jdbc:weblogic:jts:ejbPool");
}
```

Listing 3: Bean-managed Read

```
public TicketEntityPK ejbFindByPrimaryKey
  (TicketEntityPK pk)
  throws FinderException, RemoteException
{

// ejbFindByPrimaryKey fetches the Entity
// Bean

  if ((pk == null))
    throw new FinderException
      ("Invalid parameter. "+
        "Primary Key cannot be null");

      System.out.println("ejbFindByPrima
         ryKey()");

// call helper method
    read(pk);
    return pk;
}

private void read(TicketEntityPK pk)
throws RemoteException, FinderException {

// reads data from Ticket table
// into TicketEntityBean
  Connection con = null;
  PreparedStatement ps = null;
  try{
    con = getConnection();
    ps  = con.prepareStatement
    ("select * from Ticket "+
    "where ticketNum = ?");

    ps.setInt(1, pk.ticketNum);
    ps.executeQuery();
    ResultSet rs = ps.getResultSet();

// map data into Entity Bean
    if (rs.next()) {
      ticketNum = rs.getInt(1);
      price = rs.getDouble(2);
      flightNumber = rs.getInt(3);
      seatNumber = rs.getString(4);
      departDt = rs.getDate(5);
      arrivalDt = rs.getDate(6);
      departCity = rs.getString(7);
      arrivalCity = rs.getString(8);
      passengerNumber = rs.getInt(9);

      isModified = false;
    }
    else {
      throw new FinderException
        ("Read Error: TicketEntityBean "+
           pk.ticketNum + " not found");
    }
  }
  catch (SQLException sqlEx) {
           throw new RemoteException
                (sqlEx.getMessage());
  }
  finally {
    close(ps, con);
  }
}
```

Listing 4: Bean-managed Update

```
public void ejbStore()
throws RemoteException
{

// saves Entity Bean to persistent storage
  if (!isModified())
      return;

  Connection con = null;
  PreparedStatement ps = null;
  try {
    con = getConnection();
    ps = con.prepareStatement
    ("update Ticket set ticketNum = ?,"+
     "price = ?, flightNumber = ?, "+
     "seatNumber = ?, departDt = ?, "+
     "arrivalDt = ?, departCity = ?, "+
     "arrivalCity = ?, passengerNumber =
         ?"+ "where ticketNum = ?");

// fill values from TicketEntityBean
// into PreparedStatement
    ps.setInt(1, ticketNum);
    ps.setDouble(2, price);
    ps.setInt(3,flightNumber);
    ps.setString(4,seatNumber);
    ps.setDate(5, departDt);
    ps.setDate(6,arrivalDt);
    ps.setString(7,departCity);
    ps.setString(8, arrivalCity);
    ps.setInt(9,passengerNumber);
// extra for PrimaryKey where clause
    ps.setInt(10, ticketNum);

    int i = ps.executeUpdate();
    if (i == 0) {
      throw new RemoteException
      ("Failed to update TicketEntityBean:
                   " + ticketNum);
    }

    isModified = false;
  }
  catch (RemoteException ex)
  {
     throw ex;
  }
  catch (SQLException sqlEx)
  {
     throw new RemoteException
               (sqlEx.getMessage());
  }
  finally
  {
     close(ps, con);
  }
}
```

Listing 5: Bean-managed Delete

```
public void ejbRemove()
throws RemoteException
{
// deletes Entity Bean from
// persistent storage
  Connection con = null;
  PreparedStatement ps = null;
  try {
    con = getConnection();

    TicketEntityPK pk =
    (TicketEntityPK) ctx.getPrimaryKey();
    ps= con.prepareStatement
    ("delete from Ticket where ticketNum =
     ?"); ps.setInt(1, pk.ticketNum);

    int i = ps.executeUpdate();
    if (i == 0) {
      throw new RemoteException
      ("Delete failed. TicketNumber: "
      + pk.ticketNum + " not found");
    }
  }
  catch (RemoteException ex) {
    throw ex;
  }
  catch (SQLException sqlEx) {
    throw new RemoteException
(sqlEx.getMessage());
  }
  finally {
    close(ps, con);
  }
}
```

Listing 6: Bean-managed finder: ejbFindTickets By Flight()

```
public Enumeration ejbFindTicketsByFlight
  (int aFlightNumber)
  throws FinderException, RemoteException
{
  Connection con = null;
  PreparedStatement ps = null;
  ResultSet rs = null;
  try {
    con = getConnection();
    ps = con.prepareStatement
    ("select ticketNum from Ticket "+
      "where flightNumber = ?");
    ps.setInt(1, aFlightNumber);
    ps.executeQuery();
    rs = ps.getResultSet();

    Vector ret = new Vector();
    TicketEntityPK pk;

    while (rs.next())
    {
         pk =
         new TicketEntityPK(rs.getInt(1));
        ret.addElement(pk);
    }

    rs.close();
    return ret.elements();
  }
  catch (SQLException sqlEx) {
    throw new FinderException
    (sqlEx.getMessage());
  }
  finally {
    try {
      if (rs != null) rs.close();
      if (ps != null) ps.close();
      if (con!= null) con.close();
  }
    catch (Exception done) {}
  }
}
```

Listing 7: Container-managed JDBC TicketEntiyBean

```
package jdj.ticketing.containermanaged;

import java.lang.*;
import java.rmi.*;
import java.sql.Date;
import javax.ejb.*;

/*
* Example of a container-managed Entity
Bean
*/
public class TicketEntityBean extends
Object
  implements EntityBean
{
// TicketEntityBean business attributes
  public int ticketNum;
  public double price;
  public int flightNumber;
  public String seatNumber;
  public Date departDt;
  public Date arrivalDt;
  public String departCity;
  public String arrivalCity;
  public int passengerNumber;

// tells whether or not data was
// actually modified
  private boolean isModified;

// needed by Entity Bean specification
  transient protected EntityContext ctx;

// Business methods for TicketEntityBean
// Getter/Setter methods not shown in
// Listing… Methods to satifsy Entity Bean
// interface
  public void ejbCreate
    (int aTicketNum, double aPrice,
    int aFlightNumber, String aSeatNumber,
    Date aDepartDt, Date aArrivalDt,
    String aDepartCity, String
    aArrivalCity,
    int aPassengerNumber)
  {
// set attributes to insert
    ticketNum = aTicketNum;
    price = aPrice;
    flightNumber = aFlightNumber;
    seatNumber = aSeatNumber;
    departDt = aDepartDt;
    arrivalDt = aArrivalDt;
    departCity = aDepartCity;
    arrivalCity = aArrivalCity;
    passengerNumber = aPassengerNumber;

// mark as modified
    isModified = true;
  }

  public void ejbPostCreate
    (int aTicketNum, double aPrice,
    int aFlightNumber, String aSeatNumber,
    Date aDepartDt, Date aArrivalDt,
    String aDepartCity, String aArrivalCi
    ty,int aPassengerNumber)
  {
// do nothing
  }
  public void setEntityContext(EntityCon-
text aCtx)
  {
    ctx = aCtx;
  }

  public void unsetEntityContext()
  {
    ctx = null;
  }
  public void ejbRemove()
  {
// do nothing
  }

  public void ejbActivate()
  {
// do nothing
  }

  public void ejbPassivate()
  {
// do nothing
  }

  public void ejbLoad()
  {
// do nothing
  }

  public void ejbStore()
  {
// do nothing
  }
}
```

Listing 8: Container-Managed File TicketEntityBean

```
package
jdj.ticketing.containermanaged.file;

import java.lang.*;
import java.rmi.*;
import java.sql.Date;
import javax.ejb.*;

/*
*   Example of a container-managed Entity
Bean
*/
public class TicketEntityBean extends
Object
  implements EntityBean
{
// TicketEntityBean business attributes
  public int ticketNum;
  public double price;
  public int flightNumber;
  public String seatNumber;
  public Date departDt;
  public Date arrivalDt;
  public String departCity;
  public String arrivalCity;
  public int passengerNumber;

// tells whether or not data was
// actually modified
  private boolean isModified;

// needed by Entity Bean specification
  transient protected EntityContext ctx;

// Business methods for TicketEntityBean
// Getter/Setter methods not shown in
// Listing… Methods to satifsy Entity Bean
// interface
  public void ejbCreate
    (int aTicketNum, double aPrice,
    int aFlightNumber, String aSeatNumber,
    Date aDepartDt, Date aArrivalDt,
    String aDepartCity, String aArrivalCi
    ty, int aPassengerNumber)
  {
// set attributes to insert
    ticketNum = aTicketNum;
    price = aPrice;
    flightNumber = aFlightNumber;
    seatNumber = aSeatNumber;
    departDt = aDepartDt;
    arrivalDt = aArrivalDt;
    departCity = aDepartCity;
    arrivalCity = aArrivalCity;
    passengerNumber = aPassengerNumber;

// mark as modified
    isModified = true;
  }

  public void ejbPostCreate
    (int aTicketNum, double aPrice,
    int aFlightNumber, String aSeatNumber,
    Date aDepartDt, Date aArrivalDt,
    String aDepartCity, String aArrivalCi-
ty,
    int aPassengerNumber)
  {
// do nothing
  }
  public void setEntityContext(EntityCon
    text aCtx)
  {
    ctx = aCtx;
  }

  public void unsetEntityContext()
  {
    ctx = null;
  }
  public void ejbRemove()
  {
// do nothing
  }

  public void ejbActivate()
  {
// do nothing
  }

  public void ejbPassivate()
  {
// do nothing
  }

  public void ejbLoad()
  {
// do nothing
  }

  public void ejbStore()
  {
// do nothing
  }
}
```

▼▼▼▼▼▼ CODE LISTING ▼▼▼▼▼▼

Code listings for this article can also be located at www.JavaDevelopersJournal.com

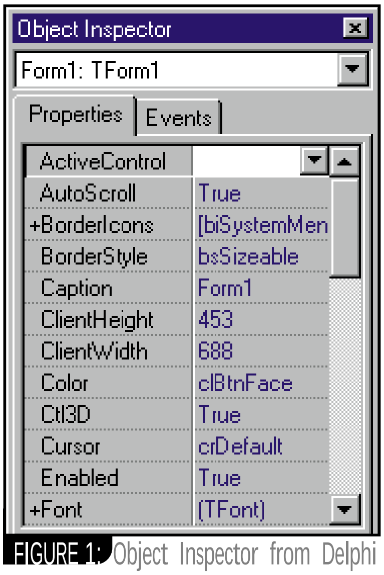
Reflecting a Bean onto a Table

Advanced JTable customization and object reflection with Java

WRITTEN BY
JIM CRAFTON

Originally I planned to continue with the syntax-highlighting CodeDocument component, but I decided to switch gears and discuss some neat uses for the JTable component that comes with Swing (my apologies go out to all those weeping in the aisles, anxiously awaiting more syntax-highlighting code...oh, just a second, let's dab the tears away before continuing).

One of the cool features of the JTable component is its ability to be customized, right down to the individual table cell. This feature, plus another unspeakably cool feature of Java itself (called *reflection*, but more on that later), will allow us to build a simple Component Inspector similar to what you'd find in IDEs like Borland's JBuilder or Symantec's VisualCafé.

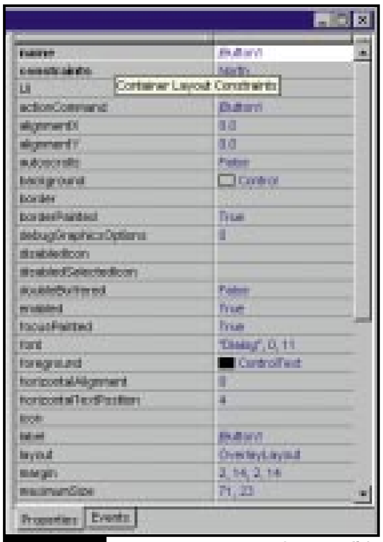


Object Inspector

Form1: TForm1

ActiveControl	
AutoScroll	True
+BorderIcons	[biSystemMen
BorderStyle	bsSizeable
Caption	Form1
ClientHeight	453
ClientWidth	688
Color	clBtnFace
Ctl3D	True
Cursor	crDefault
Enabled	True
+Font	(TFont)

FIGURE 1: Object Inspector from Delphi



name	button1
constraints	North
UI	
actionCommand	button1
alignmentX	0.0
alignmentY	0.0
autoscrolls	False
background	☐ Control
border	
borderPainted	True
debugGraphicsOptions	0
disabledIcon	
disabledSelectedIcon	
doubleBuffered	False
enabled	True
focusPainted	True
font	"Dialog", 0, 11
foreground	■ ControlText
horizontalAlignment	0
horizontalTextPosition	4
icon	
label	button1
layout	OverlayLayout
margin	2, 14, 2, 14
maximumSize	71, 23

FIGURE 2: Component Inspector from JBuilder

What Is a Component Inspector?

A Component Inspector is used to look at (and edit) the various properties that make a GUI control, like a text box or a progress bar. The Inspector is activated by selecting a control, then editing the properties through the Inspector. This idea is used not only in Java IDEs, but also in programming tools like Visual Basic and Borland's Delphi (see Figures 1 and 2.).

Our Component Inspector will consist of a JFrame, which will house a table with two columns, and a couple of labels above the table, to tell us general information about the component we're looking at. What we should end up with can be seen in Figure 3.

Now, making a super-sophisticated Component Inspector would be really cool, but the whole point of this article is to learn more about JTables and reflection, so to test this out we'll just have another window that contains a variety of controls, and each time we click on one of them the Component Inspector will be updated. Take a look at Figure 4 to see what this window looks like.

Tables, Tables . . . and More Tables

Like the rest of Swing, JTable uses the Model View Controller architecture to allow all sorts of complex customization. One type allows the programmer to specify exactly how each table cell is rendered by the table. A table cell – the individual element at the intersection of a given row and column – can be customized as to how it looks (how it renders itself for the user) and how it's edited. Not only can a JTable be customized at the cell level, but you can also write a simple table model class that can prevent columns from being edited. This is what we want to do since the left-hand column will only need to display values, not edit them. So let's just jump in.

The first thing we're going to look at is the DefaultTableModel class, which inherits from the AbstractTableModel. We could have chosen to subclass the AbstractTableModel, but we would have had to implement everything ourselves. Since the only piece of functionality we want to add at this point is to lock the leftmost column against being edited, it's much simpler to extend an already functional class – namely, DefaultTableModel. To prevent column 0 from being edited, we override the functionality of the method isCellEditable(), which is called whenever the Table Model receives notification that a cell needs to edited. If the function returns true, editing is allowed; if false, then editing is prevented. By returning false at the appropriate times, we can easily prevent column 0 from being edited. Take a look at the code in Listing 1.

The first thing we do is to add all the constructors of the DefaultTableModel so they can be called from our class. Next, we override the isCellEditable() method. In this method we check for what column is being requested. If it's 0, we automatically return false; otherwise we call the super class's isCellEditable method to handle any remaining cases.

Now let's play around a bit and learn how to customize the Table rendering. As I mentioned before, each cell that gets rendered can be customized with its own renderer, which is associated with the class type of the value in the cell. Let's say you had a cell at column 0 and row 0 that was a String object. You could associate a custom cell renderer with any String object, and the table would automatically use your renderer to paint the cell instead of its own. Not only would it paint the cell at column 0, row 0, using your renderer, but any other values in the table that were String objects would also get painted using the same renderer. Any object can be a renderer so long as it implements the TableCellRenderer interface (in the com.sun.java.swing.table package). The TableCellRenderer interface has only one method, getTableCellRendererComponent(), which needs to return a Component object. The arguments of the method can tell you whether the cell is selected or not, whether it has focus, what the value of the cell is, the column and row of the cell, and the actual table component the cell belongs to. A very simple implementation could be the one in Listing 2.

To use this, look at Listing 3.

A cell renderer can also be a component itself. For example, let's say that for boolean objects we want a checkbox to appear (there already is a cell renderer available for boolean values built into the JTable implementation, courtesy of the nice folks at Sun). We could just create a new sub class of JCheckBox and

implement the TableCellRenderer interface and we'd be all set to go. Let's look at the example in Listing 4.

One thing to remember is that the value argument of getTableCellRendererComponent() can be null, but you still have to return a component of some sort or the table will throw a sea of exceptions when it tries to draw itself, especially if you scroll down through a number of rows.

Now that we can customize our display of call data, if we were going to allow editing, we'd want to create a cell editor now as well. In this example, however, we're just going to try and display information only.

Mirror, Mirror on the Wall...

Okay, so we know how to display our data, but where do we get it from and how do we get at it? This is accomplished using a very cool – and very powerful – feature of Java called *reflection* (also sometimes referred to as *class* or *JavaBean introspection*). What you read here will be used not only in this article, but again when we come back to our CodeDocument class and start adding features like dropping down a listbox full of the variable's methods and attributes just typed in.

Reflection allows the programmer to ask an object to describe itself by listing all of its methods, fields and method signatures. It is literally like walking up to a person and asking them to describe themselves for you, tell you their family history (who their parents were, where they were born, etc.), list all the things they can do and so on. Like reflection, JavaBeans introspection allows you to ask for all sorts of detailed information pertaining to the specific instance of a bean at runtime, which is what allows tools like JBuilder and VisualCafé to list all the properties of a particular bean.

Reflection starts by obtaining a reference to the object's Class attribute. This is accomplished by the getClass() method in the Object base class. Once you have a Class object, you can get the rest of the information you need – and can even call the methods you retrieve! Let's look at Listing 5 for an example.

The first thing we do in the code is to get the Class object from the argument aValue. Now remember, because we've defined the aValue argument as an object, it could represent anything at runtime – a JFrame, a Hashtable or anything else. Once we have a reference to the class object, we can get the class name through the getName() method, which returns a fully qualified name (i.e., instead of "String", it returns "java.lang.String"). Any of the constructors, methods or fields of the class can be retrieved through calls like getDeclaredMethod() (for a single method) or getDeclaredMethods() (for all of the methods) (for constructors, you would use getDeclaredConstructors(), and so on). Using the getDeclaredMethods() method, we can loop through all the methods and print out their names using the Method class's getName() function. When retrieving a single method, you have to pass in the method name as well as an array of class objects that represent the arguments of the method. So let's say we were going to try and find out if the object had a setElementAt() method. We could do this as shown in Listing 6.

The array of class objects tells Java what types are in the argument list. If there are no arguments, you can just pass in null. Invoking the method is similar to retrieving it. You pass in an array of objects that are the actual values you want passed as arguments to the method. That would be as in Listing 7.

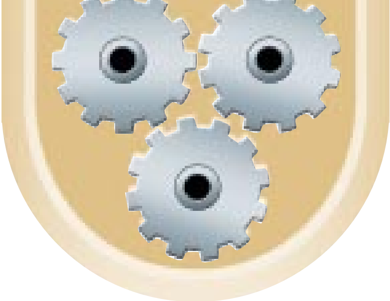
Presto! You have magically invoked a method, determining everything at runtime!

Getting BeanInfo

As I mentioned earlier, if the object you're dealing with is a JavaBean, you can get even more information – things like what kind of property editors it uses, what the display name is, whether a particular property of the JavaBean knows how to paint itself, and many others.

To start, you use the Introspector static class method getBeanInfo(), passing in the Class object of the bean or component you're interested in. The Introspector is nice enough to package everything in a neat interface called BeanInfo that has a number of methods to retrieve things like the icon associated with the bean, as well as all the properties, events and methods for that bean. The item we're really interested in is the list of properties, retrieved through a call to the getPropertyDescriptors() method of the BeanInfo class. This method gives us an array of PropertyDescriptors from which we can get information like the "read" and the "write" methods of the property, the property editor class (if one exists) and other information as well. Another short example of using this is the code in Listing 8.

All this code does is output the available properties with their names, and read and write method names to the command line. If you were interested in finding out what was in any of the properties, you could use the getReadMethod() function and invoke the read method as described earlier.



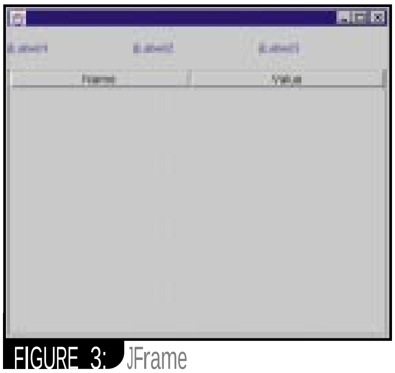


FIGURE 3: JFrame

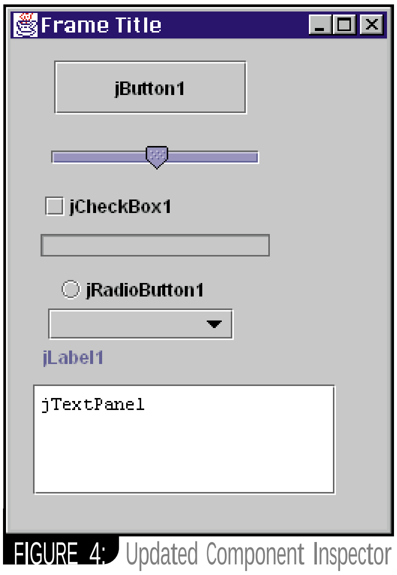


FIGURE 4: Updated Component Inspector

Tune in next time...

That's it for this month. We'll wrap it up in the next article by combining the two features we discussed: the customization of tables with reflection and JavaBean introspection, and we'll have ourselves a bona fide Component Inspector. In doing this, we'll delve further into the PropertyInspector class, learning how to paint the property and how to display any custom editors that exist for the property. Hope you found this as fascinating as I did writing it! See you next time....

About Bio

Jim Crafton is a staff consultant with Computer Sciences Corporation where he specializes in object-oriented development. He also develops advanced graphics software for Windows and the BeOS. Jim has a Web site at www.one.net/~ddiego/.

ddiego@one.net

Listing 1

```
import com.sun.java.swing.*;
import com.sun.java.swing.table.*;
import java.util.*;

public class ComponentInspectorTableModel extends Default-
TableModel{

public ComponentInspectorTableModel(){
  super();
  }

public ComponentInspectorTableModel(int numColumns,
int numRows){
    super(numColumns, numRows);
  }

public ComponentInspectorTableModel(Object[] columns,
int numRows){
    super(columns, numRows);
  }

public ComponentInspectorTableModel(Object[][] columns,
Object[] rows){
    super(columns, rows);
  }

public ComponentInspectorTableModel(Vector columns,
int numRows){
    super(columns, numRows);
  }

public ComponentInspectorTableModel(Vector columns,
Vector rows){
    super(columns, rows);
  }

public boolean isCellEditable(int row, int column){
    if (column ==0){
      return false;
    }
    else{
      return super.isCellEditable(row, column);
    }
  }

}
```

Listing 2

```
public class SimpleValueRenderer implements
TableCellRenderer{
  Jpanel cell = new Jpanel();
  Jlabel label = new Jlabel();

  public SimpleValueRenderer (){
    cell.setBackground(Color.darkGray);
    cell.add(label);
  }

public Component getTableCellRendererComponent(JTable
table, Object value,
                           boolean isSelected,
                           boolean hasFocus,
                           int row, int col){
    label.setText(value.toString());
    return cell;
  }
}
```

Listing 3

```
JTable aTable = new JTable
..
..

//previous initialization code...
  aTable.setDefaultCellRenderer(String.class,new
  SimpleValueRenderer());
```

Listing 4

```
public class BooleanValueRenderer extends JCheckBox
implements TableCellRenderer{
  public BooleanValueRenderer (){
    super();
  }

public Component getTableCellRendererComponent(JTable table,
Object value,
                           boolean isSelected,
                           boolean hasFocus,
                           int row, int col){
    this.SetValue((Boolean)value);
    return this;
  }
}

and to use it...

..
..

//previous initialization code...
  aTable.setDefaultCellRenderer(Boolean.class,new
  BooleanValueRenderer ());
```

Listing 5

```
public void getClassInfo(Object aValue){

  Class valueClass = aValue.getClass();

//lets write out the name of the class to command line
  System.out.println(valueClass.getName());

//lets get all the methods and then print out their names
//to command line
  Method[] classMethods = valueClass.getDeclaredMethods();

  for (int i=0; i < classMethods.length;i++){
    Method aMethod = classMethods[i];
//print out the name
    System.out.println(aMethod.getName());
  }

}
```

Listing 6

```
Class[] args = new Class[2];
args[0] = Object.class;
args[1] = Integer.TYPE;
//you have to do this because the method takes an int
//not an Integer
try{
  Method setElementAtMeth =
valueClass.getDelaredMethod("setElementAt", args);
}
catch (Exception ex){
  ex.printStackTrace();
}
```

Listing 7

```
Object[] vals = new Object[2];
vals [0] = new String("Hello there");
vals [1] = new Integer(0);
//Java will correctly convert this to an int

try{
 setElementAtMeth.invoke(aValue, vals);
//method gets invoked !
}
catch (Exception ex){
  ex.printStackTrace();
}
```

Listing 8

```
public void inspectAButton(Jbutton aBtn){
  try{
    BeanInfo beanInfo = Introspector.getBeanInfo(aBtn);
    PropertyDescriptor[] props = beanInfo.getPropertyDescrip
    tors();
     if (props != null){
       for (int i=0; i < props.length; i++){
        PropertyDescriptor pd = props[i];
        System.out.println("Property " + pd.getDisplayName()
        + " has read method: " + pd.getReadMethod().get
        Name() + " and write method: " +
        pd.getWriteMethod().getName());
       }
     }
  }
  catch(Exception ex){
    ex.printStackTrace();
  }
}
```

▼▼▼▼▼▼ CODE LISTING ▼▼▼▼▼▼

The code listing for all articles can also be located at
www.JavaDevelopersJournal.com

JDJ's Exclusive JavaOne Coverage

WRITTEN BY ALAN WILLIAMSON

Where were you in mid-June 1999, between the 15th and the 18th? I know where at least 20,000 of you were: Moscone Center, San Francisco.

San Francisco was host to this year's JavaOne conference – the ultimate show for anyone involved in the Java universe. If you didn't manage to make it out to the West Coast, lend me your eyes for a wee while and I'll take you through some of the things you missed. If you did manage to go, let me hopefully jog some happy memories as I take you behind the scenes.

Most people's JavaOne experience begins with the queue for their regulation JavaOne backpack – which this year was a much more functional beast than the black bag we got last year. So, with our newfound friend for the week strapped securely to our shoulder, we descended into the belly of the Moscone Center.

Last year, the hall to the left of the escalators was the main exhibitor hall, which shared its real estate with the café. This year the hall was allocated entirely to café space, which gave a hint that this year's JavaOne was going to eclipse the previous one.

As I wandered about the empty halls of Moscone, I was thinking about how, in a matter of a couple of hours, these halls would be filled with thousands of people involved with Java at all levels.

Some of the Highlights

Historically, there are only two real keynotes you simply have to attend: the opening keynote and the one Scott McNealy delivers. This isn't to say that the rest of them are boring or uninteresting, just optional.

John Gage opened the first keynote with an array of rather impressive statistics. For example, this year's JavaOne played host to over 800 speakers to inform and entertain what was expected to be over 20,000 people. This attendance figure was indeed reached. When you compare this to the first-ever JavaOne of just a few years ago, when there were just 6,000 attendees, the growth rate is astonishing.

John Gage did his usual show introduction and let us in on some of the exciting things that would be coming our way over the week. Last year, the whole conference went fractal mad, as we all clambered to get our Java ring into the picture. This year we had a new toy: the PalmPilot V, one of the first handheld devices to carry the Java Virtual Machine. It allowed us to beam class files to one another, which made for a rather interesting game that John Gage set up to ensure that everyone at the conference interacted with everyone else.

Sun wasn't as generous as they had been with the Java rings, however: we had to actually buy the PalmPilot as it wasn't part of our wonderful backpack. That said, 3Com sold them at a highly reduced price, which made it extremely cost effective. I didn't succumb to this opportunity as my view of these devices is that they're purely a novelty. One wonders how many of them will still be in use after a month. Maybe I'm cynical, but hey, that's what I'm here for.

One of the other handhelds shown at JavaOne was the Motorola pager, which also housed the JVM. It was said that the pager was also considered to be the gimmick of the show, but the FCC wasn't happy to have 20,000 radio beacons in one very concentrated area. Either way, the ability to beam Java classes – for example, a game – between these differing platforms served only to reinforce the cross-platform features of Java. It was indeed a wonderful demonstration. It's one thing to have the same class run in both an Apple environment and a Windows environment, but to see it work with devices that are so different, such as the PalmPilot and a pager, is indeed a sight to behold.

Alan Baratz was up next, and after we were introduced to his daughter in a Linux-Java jive, he went on to announce a number of new developments that will affect us all.

As we know, Java is growing at a tremendous pace. The speed at which APIs are being added makes even the most competent of Java developers worry about being left behind. But fear not. This explosion of class files has now been given a structure, and it was this new grouping that Alan Baratz introduced us to.

Java is now grouped into three editions: Enterprise, Standard and Micro. This is how Java 2.0 will now be referred to. You can learn more about the new editions from the main Java Web site, but don't worry – no new APIs have been added, just logically grouped.

Wherever you went, there was no denying it was a Sun conference. Apart from the Sun logo everywhere, the army of black T-shirts that wove in and out of the delegates – rushing back and forth between the main exhibit hall and the birds-of-a-feather sessions – were a testament to the fact that Sun had lost most of its workforce for those four days.

Speaking of BOFs, this year's offering explored every single aspect of the Java world. No matter what area of Java you were interested in, a BOF existed that allowed you to get closer to the engineers driving many of the APIs. This was good on a number of levels, as many of the attendees contribute to the various API mailing lists, and it was a great opportunity to be putting faces to the names that regularly invade inboxes. One complaint was that many of the sessions overflowed with people. This is a good sign in a way, as it highlights the popularity of these types of sessions. It will be interesting to see how Sun addresses this demand for knowledge next year.

Down in the main exhibitor hall an explosion of companies displayed their wares. If you remember, last year Sun took center stage, with all other companies on the periphery. The main problem with this was that the majority of delegates went to the Sun stand only and never ventured elsewhere. This year, Sun pulled a rabbit out of the hat by placing itself all around the outside edges and having the center filled with all third-party companies. This circulating technique worked wonderfully well, with many of the exhibitors I spoke to pleased with the amount of traffic flowing past.

As usual, Sun had a booth for each major API, manned by the actual chaps behind said API. This proved to be successful as it gave delegates the opportunity to quickly hone in on the particular area they wanted information on. I think this is a wonderful way to bring the backroom developer out into the fresh air once in a while. I've never been to a Microsoft conference, so I have no idea if Microsoft is brave enough to wheel out the poor team working on NT or 98.

We had our radio booth up in the media hall, and over the course of the four days we conducted well over 40 radio interviews. We had all the big names from Sun, including James Gosling, George Paolini and Bill Roth. In addition to this, a number of companies took the show opportunity to announce key press releases. We had them all. So drop by the main ***JDJ*** Web site to catch up on the latest press, and listen to our own Chad "voice of CNN" Sitler. What a voice that man has!

JavaOne 1999 was indeed a fantastic conference – probably the best I've ever attended, and I'm including non-Java ones in this statement. It had everything a Java developer could have wished for. Of course, there were things that could have been done better; there always are. But on the whole, I know I got a lot out of it, as did many of you that spoke to me about the whole show.

JavaOne 2000 will be held in the Moscone Center between June 5 and 8. This scares me. If the growth rate continues, then we'll simply have to ask the residents of San Francisco to move out for four days, and leave the lights on.

Until then, back to development I go.

Java Developer's Journal Readers' Choice Awards, the "Oscars of the Software Industry ," Recognized 14 Winners and

Java Developer's Journal, the world's leading publication targeting Java professionals, recognized the best software products providing business solutions with Java. ***JDJ*** presented its Readers' Choice Awards during a ceremony held at JavaOne. The awards were given to the winners and finalists before a crowd gathered at the SYS-CON Radio booth where JavaDevelopersJournal.com and SYS-CON Radio brought this event live to the software industry and the readers of ***JDJ***.

More than 15,000 ***JDJ*** readers cast their votes to select the best products of the year. The polls opened on February 1, 1999, on our Web site and ran through May 15, 1999. The votes were tabulated and audited for accuracy by BPA International, the largest international audit firm specializing in magazine industry circulation audits. The results of the poll were received at our offices on June 6, 1999. Winners and finalists in 14 award categories, and CNN.com, which won the Editor's Choice for "Best Online Technology Coverage," were acknowledged for the contributions they have made in developing Java-based solutions that respond to and meet the increasing demands of the industry.

JDJ presented the Readers' Choice Award winning and finalist products

Best Java IDE

Best Application Server

Best Java Class Library

Best Java Middleware

Best JavaBean

Best Java Installation Tool

Best Java Modeling Tool

Best Java Profiling Tool

Best Java Reporting Tool

Best Team Development Tool

Best Java Testing Tool

Best Java Application

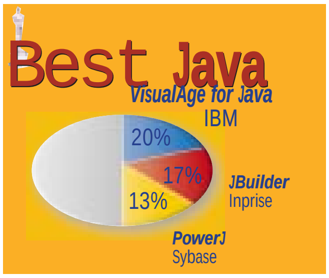
Most Innovative Java Prod-

"Products that advance the age of Java technology are introduced every day," said Sean Rhody, ***JDJ***'s editor-in-chief. "The tremendous amount of product nominations we received created a great challenge for the participating vendors of the 1999 Readers' Choice Awards. After a close review of all the nominations, the winners and finalists did not come as a surprise."

JDJ's Readers' Choice Awards, also referred to as the "Oscars of the software industry," are given to the best Java products of the year. These products are nominated by their vendors, public relations agents, users of the products, ***JDJ*** readers, developers and other members of the software community.

What's Unique About these Awards

Awards are given as a result of SYS-CON Publications' uniquely implemented voting procedure, which is openly tabulated and audited for accuracy by an independent audit firm. The results of the vote distribution could closely simulate the market share and popularity of each product as a result of wide participation by industry professionals who are among ***JDJ*** readers.



IBM
Winner: VisualAge for Java

IBM's VisualAge for Java provides advanced support for building Web-enabled enterprise applications, JavaBeans, Enterprise JavaBeans, servlets and applets. It's the only Java development environment that supports the development of Java applications that can scale from embedded systems to OS/390 enterprise application servers.

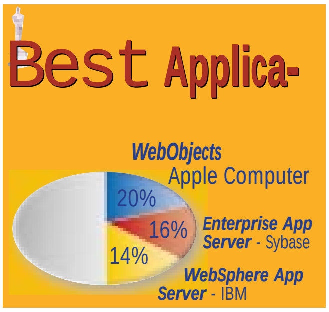
VisualAge for Java is designed to simplify Java application development by providing a visual development environment that enables programmers to assemble applications using prebuilt components such as dynamic, and to aid abstract development, efficient team development and the integration of legacy applications with Web-based environments.

Inprise
1st Finalist: JBuilder

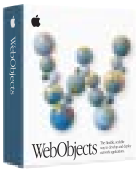
JBuilder 3 Enterprise is a comprehensive set of visual development tools for creating pure Java enterprise-scale applications for the Java 2 platform. It rapidly delivers distributed Java 2 applications using the integrated VisiBroker CORBA ORB, multitier Application Generator, BeansExpress for Enterprise JavaBeans, Java 2 remote debugger and PVCS Team Development version manager. JBuilder visually creates IDL interfaces for CORBA applications using the DataModeler and provides a graphical view of ORB services from within JBuilder using the ORB Explorer.

Sybase, Inc.
2nd Finalist: PowerJ

PowerJ provides an end-to-end solution for building Internet applications, exploiting the benefits of HTML and Java clients. PowerJ offers database capabilities while integrating seamlessly with Sybase Enterprise Application Server, enabling enterprise-class applications from creation through testing and debugging to deployment. PowerJ features comprehensive support for Java standards, thin-client deployment including an HTML editor, an integrated multi-language script editor, a repository for version control, a development version of Sybase Enterprise Application Server that allows the user to deploy and debug components without leaving the development environment, and Java and HTML DataWindows.



Apple Computer
Winner: WebObjects



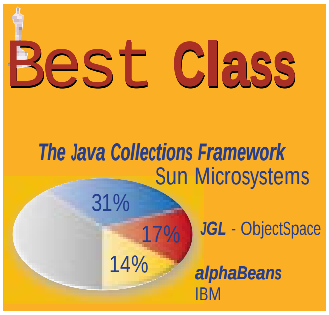
WebObjects, application server software for developing and deploying high-performance, large-scale Internet and intranet sites, combines Apple's strength in Web-content creation with performance productivity and integration. Running on UNIX, NT, Mac OS X and HP-UX, WebObjects provides an environment for enterprise applications to work across system, network and application boundaries. With WebObjects, developers can build applications that can be scaled from small workgroups to millions of users. WebObjects comes with an integrated suite of development tools and deployment capabilities for building distributed Java applications.

Sybase, Inc.
1st Finalist: Enterprise Application Server

A highly scalable, robust deployment foundation for Web and distributed applications, Sybase Enterprise Application Server incorporates the capabilities of a component transaction server and a dynamic Web page server. EAServer offers cross-client and cross-component support for almost any type of distributed application – those based on CORBA, JavaBeans, Enterprise JavaBeans, PowerBuilder and COM components, as well as existing C/C++ applications – and provides a single point of integration for heterogeneous back-office systems and extends customers' businesses to the Web.

IBM
2nd Finalist: WebSphere Application Server

The IBM WebSphere Application Server represents a complete range of Web application server environments supporting business applications from simple Web publishing through enterprise-scale transaction processing.



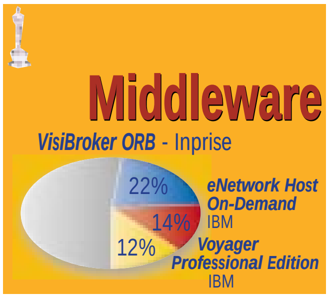
Sun Microsystems
Winner: The Java Collections Framework

The Java Collections Framework, a part of the Java 2 platform, is a unified architecture for representing and manipulating collections of objects. The Framework allows collections to be manipulated independently and reduces programming effort while increasing performance. It allows for interoperability among unrelated APIs, reduces effort in designing and learning new APIs, and fosters software reuse.

ObjectSpace Inc.
1st Finalist: JGL

ObjectSpace JGL is a Java adaptation of the ANSI/ISO standard template library that extends the JDK with a series of 11 advanced collections and more than 50 generic algorithms. JGL includes full source code, hundreds of examples, and a comprehensive online HTML tutorial and class reference. JGL also offers distributed collection support, allowing the remote construction, access and persistence of all JGL containers using ObjectSpace Voyager, the standards-neutral platform for object computing.

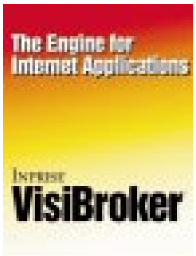
IBM



2nd Finalist: alphaBeans

See **Best JavaBean Winner**

INPRISE
Winner: VisiBroker ORB



VisiBroker provides the infrastructure necessary to enable industry-standard, cross-platform communication among distributed objects. VisiBroker provides a complete CORBA ORB environment for building, deploying and managing distributed Java and C++ objects. With native implementations of the Internet Inter-ORB Protocol (IIOP), VisiBroker allows you to take advantage of the opportunities presented by Web, Internet and intranet-based technologies while leveraging the component reuse fostered by object-oriented computing.

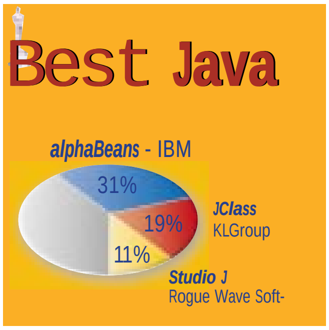
IBM
1st Finalist: eNetwork Host On-Demand

IBM eNetwork Host On-Demand provides easy and secure host access to users in Internet-based environments, including intranets and extranets. It enables businesses to extend the reach of their host applications and data to new users, including business partners, suppliers and sales personnel.

Host On-Demand gives users secure browser access to host applications and data with Java-based emulation. With support for TN3270E, TN5250, VT52/100/220 and CICS applications included in a single package, users need only one interface to reach key host data.

ObjectSpace, Inc.
2nd Finalist: Voyager ORB Professional

Voyager ORB Professional, formerly Voyager Professional Edition, is a standards-neutral, 100% Pure Java object request broker that simultaneously supports CORBA, RMI and, soon, DCOM. Voyager ORB Professional includes a graphical management console, configuration framework, JNDI integration, persistent



directory and CORBA naming service and support for ultra-light.

IBM

Winner: alphaBeans

alphaBeans include a variety of JavaBeans that can be used to develop Java applets or applications. Use these beans in any visual builder tool, such as IBM VisualAge for Java, to quickly and easily build Java programs through visual wiring – without writing a line of code. Currently, alphaBeans include more than 300 beans in areas of user interface, XML, wiring helpers, networking, masking, image processing and much more.

A unique feature of alphaBeans is their wireability; they can be used in an IDE by simple visual programming – even if you don't know Java. Recently released beans also include special "helpful" features to make them even easier to use in visual programming environments.

KL Group Inc.

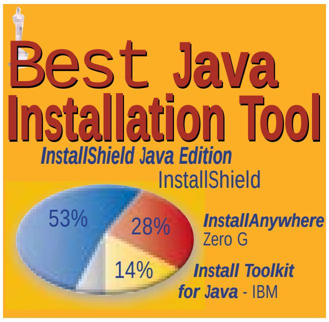
1st Finalist: JClass Enterprise Suite

The JClass Enterprise Suite is a comprehensive collection of JavaBeans, components and utilities for professional Java GUI developers. This package provides a wide range of high-value GUI functionality including charting and graphing, tables and grids, data connectivity, data input validation, JAR building, and a set of extensions and enhancements to the Swing toolkit. JClass Enterprise Suite includes a comprehensive support service and the following products: JClass Chart, JClass LiveTable, JClass Field, JClass SwingSuite, JClass HiGrid, JClass DataSource and JClass JarMaster.

Rogue Wave Software, Inc.

2nd Finalist: StudioJ Class

StudioJ offers high-quality, integrated, pure Java components for visualizing enterprise data. Its data interface allows the user to access data from a variety of sources. StudioJ allows the user to read in the data, populate the data model and then view data in a grid, chart or complex overlay charts. StudioJ also provides a number of UI components that can be used to enhance JFC Swing or AWT components.



InstallShield

Winner: InstallShield Java Edition

InstallShield Java Edition 2.5 is a tool that produces bulletproof, cross-platform installations. InstallShield walks the user through installation production, creating a single package that can be read by Java Virtual Machines version 1.1.6 or higher. Multiple source and target directory support lets the user specify more than one source directory and one target directory for installation files. InstallShield generates the applet and Web page, as well as the necessary code to allow Web users to download and launch the installation. It also has several internationalization features that allow the user to write multiple language installs so the end user can choose the language in which the installation will run.

ZeroG Software, Inc.

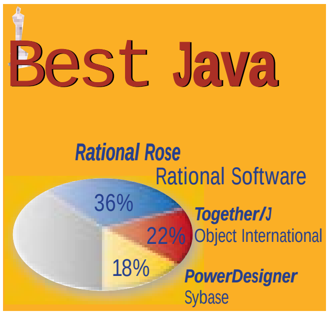
1st Finalist: InstallAnywhere

Written entirely in Java, InstallAnywhere from ZeroG Software is a powerful solution available for building professional multiplatform installers capable of installing to virtually any client or server platform. Every installer built with InstallAnywhere recognizes the platform under which it's operating and tailors its installation to the user's system.

IBM

2nd Finalist: Install Toolkit for Java

Install Toolkit for Java is a program for writing Java and non-Java install programs for OS/2, OS/390, AIX, Solaris, Linux and Windows NT/95 software. Since Install Toolkit for Java is written in Java, it can be run on any platform and operating system that supports Java. This portability provides a simpler installation and distribution process. Installing your program with Install Toolkit for Java requires one file (install.class) and easy steps for implementation. When install.class is run, the rest of the install program files are extracted from install.class. One of these files is data.zip, which contains all the files in your product to be copied during the install process, and enables the install image for your product to be built with a simple zip utility.



Rational Software

Winner: Rational Rose

Rational Rose 98i provides Unified Modeling Language-based modeling for designing component-based applications. It features multilanguage capabilities and enterprise team development features. Rose 98i Enterprise Edition has multilanguage support, allowing the user to mix and match multiple languages within the same model, and it supports C++, Java, Smalltalk and Ada, as well as 4GLs such as Visual Basic, PowerBuilder and Forte. For Java development, Rational Rose supports the design, modeling and visualization of all Java constructs.

Object International, Inc.

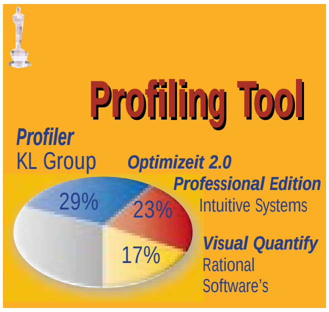
1st Finalist: Together/J

Together/J is the visual UML modeler featuring simultaneous round-trip engineering. The Whiteboard Edition features simultaneous design-and-code editing, supporting the way designers and programmers work. The Whiteboard Edition features UML class modeling developed for solo programmers building Java applications. For a limited time, developers can download it from www.togetherj.com. The Developer and Enterprise editions feature a suite of UML diagrams, custom forward and reverse engineering, HTML doc gen, version-control links and support for both Java and C++ apps.

Sybase, Inc.

2nd Finalist: PowerDesigner

PowerDesigner offers a comprehensive modeling solution that business and systems analysts, designers, DBAs and developers can tailor to meet their specific needs -- depending on the size and scope of the project. With PowerDesigner, users can create conceptual data models, then automatically generate physical data models and de-normalized physical designs for over 30 database management systems. Reverse engineering allows a designer to "blueprint" an existing database structure for documentation or retargeting to a different database. And PowerDesigner modules support all the leading development tools.



KL Group Inc.

Winner: JProbe Profiler

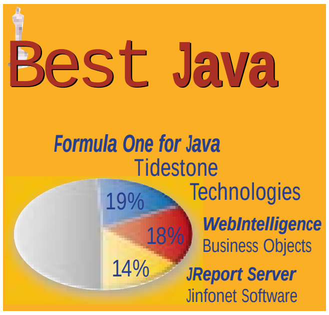
JProbe Profiler is a Java development tool that helps developers improve the performance of Java applications. JProbe Profiler lets the developer pinpoint and eliminate performance bottlenecks caused by inefficient objects in Java code. JProbe Profiler combines a visual call-graph interface and unique data collection technology with nine different performance metrics to provide accurate diagnostics on both performance and memory usage. JProbe Profiler includes the fully integrated JProbe Memory Debugger that helps developers locate memory leaks and reduce the memory used by Java applications.

Intuitive Systems, Inc.

1st Finalist: Optimizeit 3.0 Professional

Optimizeit delivers a solution for Java developers looking to track down and fix performance issues in any Java applications, servlets, applets or JavaBeans. Optimizeit 3.0 Professional offers complete profiling capabilities including memory-leak debugging, CPU profiling and monitoring of object allocations. Using Optimizeit, Java developers get complete information about CPU and memory usage on any part of their programs down to the responsible line of source code. Optimizeit allows for remote profiling of any Java process, integrates with multiple IDEs, supports most JDK 1.1.x and Java 2 virtual machines with no modifications, and is available for multiple platforms (Windows, Solaris, Linux).

Rational Software



2nd Finalist: Visual Quantify

Rational Visual Quantify is a performance profiling tool that automatically pinpoints application performance bottlenecks, taking the difficulty and guesswork out of performance tuning. Visual Quantify also delivers repeatable timing data for all parts of an application including components.

TidestoneTechnologies

Winner: Formula One for Java

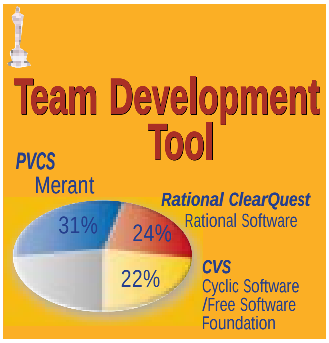
Formula One for Java is lightweight, high-performance spreadsheet technology designed for Web-based, distributed computing environments. Positioned for both server-side and client-side solutions, Formula One can be utilized as a JavaBean in larger Java applications and application servers, an applet in Web pages and a standalone application on any desktop. Formula One provides 100% Pure Java, Microsoft Excel-compatible spreadsheet functionality including a reporting tool for developers and a consequent front-end analytical/calculation tool for end users. Its multithreaded architecture allows users to distribute and receive data from multiple sources concurrently. Formula One can read, write and distribute data in .vts, .xls and HTML file formats. With its workbook designer, Formula One offers all of the data-formatting features found in typical desktop spreadsheet applications.

Business Objects

1st Finalist: WebIntelligence

WebIntelligence is a multitier, thin-client decision support system (DSS) that provides nontechnical end users with ad hoc query, reporting and analysis of information stored in corporate data warehouses, data marts and packaged business applications over the World Wide Web. WebIntelligence 2.0 offers features for extranet deployments, an online analytical processing (OLAP) module for drilling in charts and tables, and numerous options for report creation and distribution.

Jinfonet Software, Inc.



2nd Finalist: JReport Server

JReport is a pure Java report designer and application server. With JReport, users can design reports with many features including sections, subreports, nested groups, formulas, conditional formatting, graphs/charts, crosstab, RTF text and over 160 built-in functions. Extensive APIs let you access user data source, write Java functions, define new objects and import JavaBeans. The designer provides intuitive SQL, and Report Wizards and property sheets. JReport Viewer supports table of contents, drill-down, hyperlinks and on-screen sorts.

MERANT

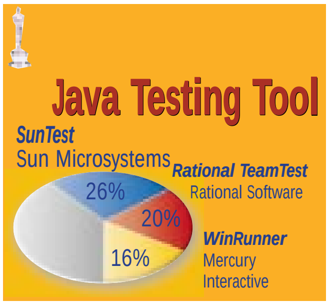
Winner: PVCS

PVCS is a comprehensive process-based configuration management tool for complex development projects allowing the user to choose the level and depth of SCM needed. As a software configuration manager (SCM), PVCS offers class tools for version management, issue and change management, and build and release management. Other features include task automation and protection against development errors. PVCS provides support for the user's choice of platforms, environments and interfaces.

Rational Software

1st Finalist: Rational ClearQuest

Rational ClearQuest is a flexible defect-tracking/change-request management system for tracking and reporting on defects and other types of change requests throughout the development lifecycle. ClearQuest provides reliable and efficient project metrics and shortens development cycles by unifying all team members – project managers, QA managers, testers and developers – in managing software development change.



C yclic Software/Free Software Foundation

2nd Finalist: CVS

CVS is a version control system that allows the user to keep old versions of files (usually source code) and a log of who, when and why changes occurred, like RCS or SCCS. CVS operates on hierarchical collections of directories consisting of version-controlled files, not just on one file or one directory at a time. CVS helps to manage releases and to control the concurrent editing of source files among multiple authors. CVS allows triggers to enable/log/control various operations and works well over a wide network.

Sun Microsystems

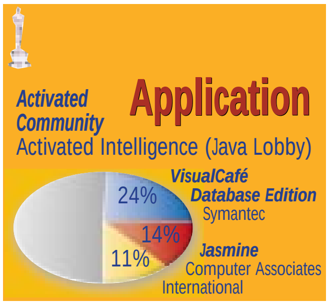
Winner: SunTest

SunTest is a suite of Java testing tools developed specifically for testing Java applications and applets. These three tools are written entirely in the Java language and are designed to work on all Java-compatible platforms. The SunTest suite consists of JavaStar, JavaSpec and JavaScope. Collectively, these tools can be used for automating Java test development and execution and for analyzing test coverage.

Rational Software

1st Finalist: Rational TeamTest

Rational TeamTest is a testing tool delivering integrated functional testing of Java, Web, ERP and client/server applications. Built on a scalable, integrated server-based test repository, Rational TeamTest combines functional testing power and comprehensive management tools for automated testing of applications. Rational TeamTest comprises the following components: a test recording tool, defect tracking, Web-site management, a Web-based defect entry tool and a test planning, management and analysis tool.



Mercury Interactive

2nd Finalist: WinRunner

WinRunner is an enterprise functional testing tool that verifies applications work as expected. By capturing and replaying user interactions automatically, WinRunner identifies defects and ensures that business processes that span across multiple applications and databases work flawlessly the first time and remain reliable throughout the lifecycle. WinRunner is integrated with all of today's leading application development and deployment environments including ERPs, the Web, Java and traditional GUI development tools.

Activated Intelligence (Java Lobby)

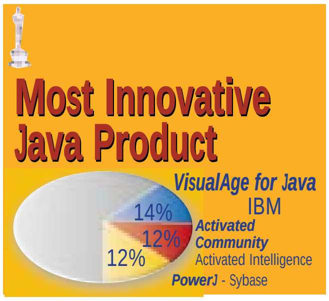
Winner: Activated Community

Activated Community, an Internet service for building sustainable online communities, is built on a pure Java foundation and uses the power of XML/XSL to enable dynamic and diverse forms of information delivery. Activated expects AC to quickly integrate with e-mail and news clients, Palm Pilots, WebTV and a host of future form factors.

Symantec Corporation

1st Finalist: VisualCafe´ Database Edition

VisualCafé Database Edition is a Java development solution for corporate databases. JavaBean, database and servlet wizards simplify and accelerate development of database-aware Java applications. VisualCafé supports such Java technology as JFC/Swing, JDBC, RMI and JDK 1.2. It includes dbANYWHERE Server for Win95/98/NT featuring JDBC/ODBC connectivity, providing native support for Oracle, Sybase, Informix and MS-SQL databases.The three-tier architecture makes it possible

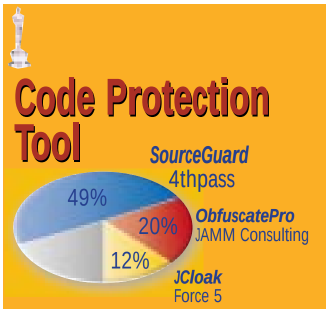


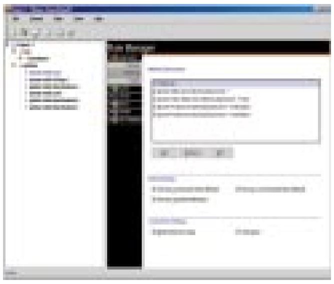
to provide lightweight applications on the client machine, offloading network and database traffic, and eliminating the need to configure and maintain large client applications.

Computer Associates International, Inc.

2nd Finalist: Jasmine

Jasmine is an industrial-strength, pure object database that features a fully integrated multimedia development environment. With Jasmine, developers can create an application once and deploy it without change across the Internet, intranet, extranet or client/server configurations. A feature of Jasmine is Jasmine Studio – a fully integrated visual application development and database management environment.





IBM

Winner: VisualAge for Java

See ***Best Java IDE Winner***

Activated Intelligence (Java Lobby)

1st Finalist: Activated Community

See ***Best Java Application Winner***

Sybase, Inc.

2nd Finalist: PowerJ

See ***Best Java IDE 2nd Finalist***

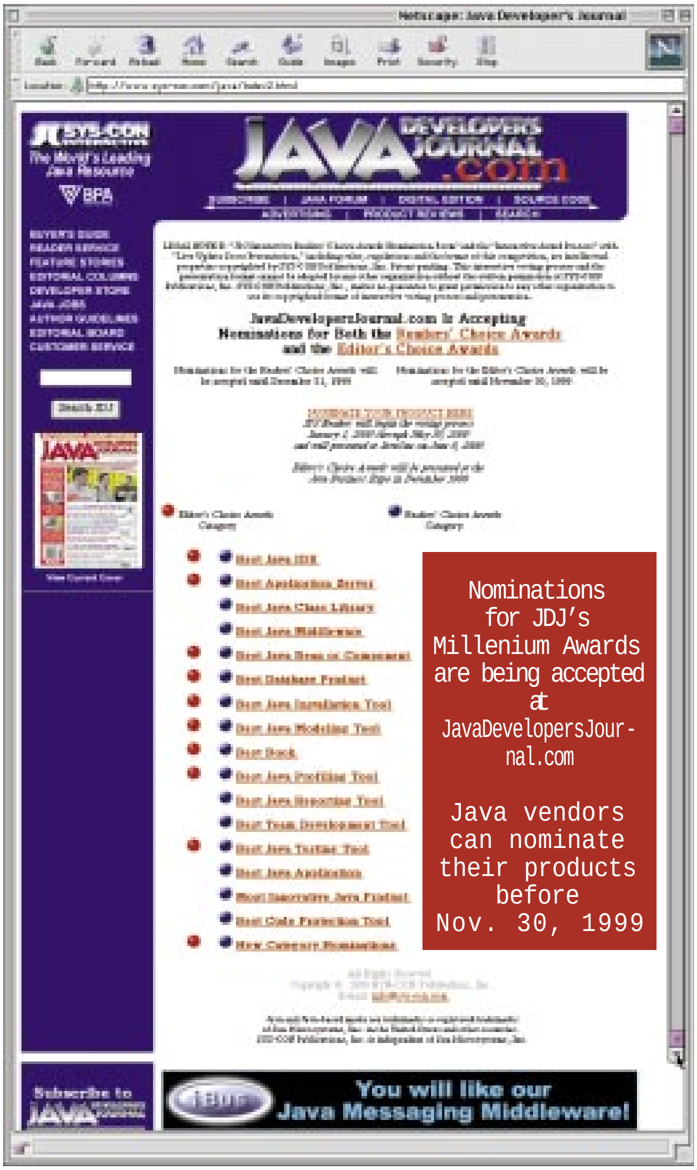
4thpass

Winner: SourceGuard

4thpass SourceGuard Enterprise Edition is a solution for protecting Java bytecode. Whether you're developing an applet, application, servlet, class library or JavaBeans, SourceGuard enables users to protect bytecode. Using Loop Hiding technology, SourceGuard transforms flow-control graphs of methods containing loops to irreducible graphs. Furthermore, SourceGuard uses Bytecode Range Modification to ensure that all methods will have a nonvalid representation in Java source code.

JAMM Consulting, Inc.

1st Finalist: ObfuscatePro



SYS-CON INTERACTIVE
The World's Leading Java Resource
BPA

JAVA DEVELOPERS JOURNAL .com

SUBSCRIBE | JAVA FORUM | DIGITAL EDITION | SOURCE CODE
ADVERTISING | PRODUCT REVIEWS | SEARCH

BUYER'S GUIDE
READER SERVICE
FEATURE STORIES
EDITORIAL COLUMNS
DEVELOPER STORE
JAVA JOBS
AUTHOR GUIDELINES
EDITORIAL BOARD
CUSTOMER SERVICE

Search JDJ

View Current Cover

JavaDevelopersJournal.com is Accepting
Nominations for Both the Readers' Choice Awards
and the Editor's Choice Awards

NOMINATE YOUR PRODUCT HERE

Editor's Choice Awards Category
Readers' Choice Awards Category

Best Java IDE
Best Application Server
Best Java Class Library
Best Java Middleware
Best Java Bean or Component
Best Database Product
Best Java Installation Tool
Best Java Modeling Tool
Best Book
Best Java Profiling Tool
Best Java Reporting Tool
Best Team Development Tool
Best Java Testing Tool
Best Java Application
Most Innovative Java Product
Best Code Protection Tool
New Category Nominations

Subscribe to JAVA

You will like our Java Messaging Middleware!

Nominations for JDJ's Millenium Awards are being accepted at JavaDevelopersJournal.com

Java vendors can nominate their products before Nov. 30, 1999

You can find detailed coverage of the 1999 Readers' Choice Awards at our Web site!

Milestones from the Poll

by Alan Williamson

February 15:

I just can't keep away from this one. Because we're representing the development community, we really can't overlook this one. To some of you, I'm sure your JBuilder or VisualCafé is your home for the best part of the day. VisualAge is still strong, but its comfort zone is not as great as it was in our last review.

March 4:

Gosh, this is a closely run race, with Apple head to head with IBM. Apple is edging out in front with their WebObjects by only 43 votes. Remember, Apple came in late to the polls with a tremendous surge. It would appear the wind in their sails is beginning to die down. However, Apple recently announced its move toward open source with its new server, so let's keep an eye on this to see if it wins over support from the development community.

An area that's close to my heart is Java servlets. We can see that the big players in the servlet world aren't faring too bad compared to the bottomless money pits of IBM and Apple. WebLogic is heading the bunch with 7% of the category's votes. Paul Colton's JRun is coming in at 6% with Stefano Mazzocchi's team's JServ coming in at 4%. These are application servers that have been designed specifically to run Java servlets, with all of the above contributing very heavily to the core design of the Servlet API.

April 1:

In the category Best Installation Tool, no real surprises are coming through here. You're voicing your support for InstallShield (49%) with InstallAnywhere (34%) coming up in an admirable second place; 41% of you voted for a product within this category. This is encouraging and very good news for Java.

April 5:

Tools to allow the development world to tune their creations to that of a well-oiled engine are included in our awards. JProbe and Optimizeit! are competing strongly for Best Java Profiling Tool. Interestingly, though, only 25% of you voted in this category. Do you think this is atypical of the number of developers that actually bother to profile their code? I sincerely hope not!

Great Products that Didn't Win!

Most of the top companies – for example, Sun, IBM, Apple – have serious marketing budgets for each of their products. But what about the smaller companies, the ones that don't have quite the same influence with journalists and find it difficult to get press?

I urge you to check some of them out.

Underdog of the IDE Category:

Elixir is sitting hear the bottom of the list of true IDEs. They have managed to pull in only a small part of the votes, and to be honest this is a bit harsh. They produce a very sexy, clean, Java-based IDE environment. It's not bloatware like many of its counterparts sitting near the top. So check them out at www.visolu.com/visprod.htm and let me know what you think.

Summary

As stated in Rick Ross's new column, launched this year, examples of small Java start-ups are already disappearing. I fully support his standpoint and wish to pick up the guantlet where Rick left it. We all need to support the smaller companies and help them by at least giving them a chance before automatically looking toward the likes of IBM or Sun for solutions.

COMPLETE CODE LISTING INCLUDED!

Back to Basics

Attention to the basics of OOP pays dividends for the creators of a new interpreter

WRITTEN BY
GENE CALLAHAN
& BRIAN CLARK

As we try to keep pace in the frantic Internet era, it's easy to become enraptured by the latest technologies – JavaBeans, CORBA, Swing and so on. However, in the rush to add the latest buzzword to our résumés or marketing brochures, we too often forget the basics of object-oriented programming. The basics were the reason OOP was developed and what first attracted us to this paradigm. How many of us have had the dismaying experience of coming across "object-oriented code" that, although it might implement a "cross-platform, event-driven, multithreaded, multitiered" application, had all the elegance and organization of spaghetti code written in unstructured Basic.

The basic principles underlying the object paradigm haven't changed since its inception. Although not all experts agree, the most common definition of an object-oriented language is that it's required to support four distinct concepts – encapsulation, data abstraction, polymorphism and inheritance. In this article we'll describe how attention to these basics has paid big dividends for us in creating the HotScheme interpreter. But first, let's review what each of these concepts means.

- ***Encapsulation***: Generally, most aspects of a class should be invisible to objects outside the class. By encapsulating member variables and methods (i.e., making them private or protected), a design can more precisely control their usage and therefore ensure the correctness of that usage.
- ***Data abstraction***: Class A, which uses class B, should be able to interact with a narrowly defined interface to B. This interface should present A with a view of B's usage, not its implementation. For example, a user of a symbol table class doesn't need to know if it's implemented as a hashtable, an array or some other concrete data structure. Users only need to know how to place a symbol in the table and how to get its value back out.
- ***Polymorphism***: Having defined an interface as above, objects that implement that interface should generally be interchangeable. If class A needs an object of class "SymbolTable" passed to its constructor, then all descendants of SymbolTable should also be acceptable parameters.
- ***Inheritance***: A class should be able to inherit as much of its behavior as possible from more primitive classes with the same signature, in order to maximize code reuse.

How These Principles Helped

When we began working on HotScheme, we didn't know where we would take the project. To test our ideas about how to build the interpreter engine (see "Design Patterns in a Java Interpreter," ***JDJ*** Vol. 4, issue 1), we started out using a simple character terminal I/O. This was the interface we had seen in most of the Lisp interpreters we had dealt with. We opened a terminal window, output a prompt and read the user's input as it occurred. As soon as we saw a closing parenthesis that matched the first opening one, we parsed and ran the user's code.

It would have been "easiest" to simply call System.out.println() and System.in.read() everywhere we needed to do I/O. However, this would have been an example of what Larry Wall, the creator of Perl, refers to as "false laziness." We would have killed off some of the work in version one only to have the ghost of that work haunt us, with an effort many times the initial savings, in versions two and three.

Fortunately, we make every effort we can to practice true laziness. Therefore, we sent all I/O through an object we called "LispTerminal." We defined a simple but sufficient interface to the class, using the time-tested metaphor of a character terminal as our model – an example of data abstraction. The class needed just three public methods – read(), print() and unread(). For the convenience of the users of the class (us!), we added a println() method (see Listing 1). We created a single, simple descendant of LispTerminal, CharLispTerminal, which sent its I/O to System.out and System.in. These actual I/O channels were hidden from users of the class through encapsulation (see Listing 2).

We decided to pass an instance of this class to methods that used it, rather than using a member variable, so that a new terminal type could be plugged in at any point while running. Here is a case of a general preference: when a private method needs to access a member variable of its own class, we like to pass it as a parameter to the method rather than access it directly. This may seem a curious waste of CPU cycles, but we've found that this practice offers in return a great deal of flexibility in using these functions, and even aids in the occasional repartitioning of a class hierarchy: if the method is directly accessing a class member, it's not easy to move it to another class.

Having completed our "proof-of-concept" cut at the interpreter, we decided we should be able to run HotScheme as an applet. This would allow an institution wanting to use HotScheme to place a single copy of the interpreter on its Web server where any student with a browser could access it. Running as an applet entailed restraints on what we could do. Besides the usual limitations imposed for security, we also had to worry about whether the JDK features we employed were generally supported in browsers.

Since we'd planned for change and extension from the start, we were able to add the GUI interface without changing the underlying interpreter. We descended another class, GUILispTerminal from LispTerminal, which had to perform some tricks that LispTerminal did not. Instead of continuously feeding input to the interpreter, GUILispTerminal forwards all input only when an "Evaluate" button is clicked. Otherwise we'd have hit conflicts between the user's ability to interact with the GUI and the interpreter's operation. For example, if we were interpreting typing as it occurred, what would we do when the user clicked the "Trace On" button in the middle of typing in a command?

Another new issue is that we now had the possibility of hitting an end-of-input condition. In character mode the interpreter simply kept trying to read a character until the user quit the application. This wouldn't work in GUI mode because we'd eventually reach the end of our input buffer. We wanted to grab control back from the interpreter and collect another batch of input. We added a single new call, eof(), and had it always return false in the base class but true in GUILispTerminal whenever the end of the buffer had been reached. A true return from eof() would break out of the interpreter's endless read-evaluate-print loop.

Since we were no longer using simple character I/O, we had to devise a replacement for our use of a PushbackInputStream in CharLispTerminal so we could "unread" a character in GUI mode as easily as we could in character mode. We had the GUI put all keystrokes into a buffer internal to GUILispTerminal. Unreading a keystroke became as simple as decrementing an index into that StringBuffer:

```
public void unread(int c) { if(pos > 0) pos--; }
```

To implement the above, we had to add a new call for use from the GUI, which we called setBuffer(). This change was transparent to the interpreter itself, however, as the call isn't needed from its vantage point. See Listing 3 for the implementation of GUILispTerminal.

Our next adventure with the LispTerminal family came when we realized that the ability to load a package of Scheme code from a URL would be useful. A site hosting HotScheme could then supply users with packages of Scheme code as libraries, sample programs and student exercises.

Our new class, URLLispTerminal, shared the need for buffering with GUILispTerminal, so we created a new common ancestor for them, BufferedLispTerminal, and moved the methods and members that enabled buffering into that class (see Listing 4). The public methods that moved were eof(), read(), unread() and setBuffer(). Using inheritance, we were then able to make use of this code in both GUILispTerminal and URLLispTerminal.

The constructor for a URLLispTerminal takes a URL and another LispTerminal as arguments. Input is fetched from the URL, while all output is merely passed through to the contained terminal that was passed in the constructor. To load and run a file of Scheme code, all we had to do was pass a new instance of URLLispTerminal to our existing function, LispInterpreter.read_eval_print_loop(). This was done in the simple functor that implements the load command. The functor itself is essentially one line of code surrounded by some exception handling:

```
// the StringVal of the first arg is the URL,
// env.getTerm will return the current LispTerminal,
// and env is essentially the current symbol table
LispInterpreter.read_eval_print_loop(
    new URLLispTerminal(
        args.first().StringVal(), env.getTerm()),
        env);
```

Although adding the ability to read code from a file based on user-typed input, adding the definitions in the file to the current environment, running the code and then returning control to the user might seem like a job that would involve changes thoughout the program, no code outside of the LispTerminal package and the Load functor changed.

We decided that we'd hit on a mechanism that would be generally useful, and abstracted it for all LispTerminals. In the base class we added two member variables, one to hold an input LispTerminal, the other an output LispTerminal. We redefined the base class implementations for reading and writing so that they'd first check the appropriate member. If it's not null, they delegate the I/O task to the contained object. This change was completely contained within the LispTerminal hierarchy. We'll illustrate the pattern with print() – read(), unread() and eof() are all similar:

```
public final void print(Object obj)
{
    if(out_term != null) out_term.print(obj);
// polymorphism – the else clause will call the descendant's
my_print()
    else
my_print(obj);
}
```

When we implemented our "plug-and-play" terminals, it occurred to us that we simply might have designed the interpreter to use separate I/O channels and done away with the terminal concept. This method would have some advantages. There'd be no need for a forwarding mechanism in LispTerminal or, indeed, any kind of terminal class at all. Everywhere we passed a LispTerminal, we would instead pass two parameters: an input sink and an output sink. However, we decided we liked the compactness of a single terminal object and the implication that both sides of I/O had to be handled as a unit. After all, in most cases the nature of the input will be tied to the nature of the output. Moreover, without the terminal object to keep track of the I/O delegation, some other mechanism would need to be created to do so. Having the LispTerminal class keeps this code all in one spot.

We're convinced that we can now easily create new terminals that combine socket, GUI, terminal and file I/O in any arbitrary combination desired. We could, for instance, create a terminal that teed its output, perhaps writing to both the screen and a file, or we could send code off to a fast server for processing.

Our design was far from ideal when we started coding. With perfect foresight, the addition of the delegation mechanism and the buffered class would have been unnecessary – we would have included them from day one. But only gods produce perfect designs. Because we followed the four principles of object-oriented design, none of the changes we had to make were onerous, and each took less than a day's worth of coding.

Our next step in the HotScheme project is to make SchemeObject a bean. Scheme differs from Java in that even the most primitive constructs in the language, such as "if" and "case," can be considered Scheme objects in their own right. In our implementation this is represented by having all of them descend from the base class SchemeObject. We intend to implement a JavaBeans interface to all of the language constructs in Scheme. This will permit the construction of visu-

AUTHOR BIOS

Gene Callahan is president of St. George Technologies, where he designs Internet projects. He has written articles for Computer Language, Software Development and Web Techniques, among others.

Brian Clark is a software engineer residing in Virginia. His current focus is on the application of design patterns on UI and middle-tier design using Java.

al editors for the language itself. Rather than presenting an abstract graphical model, then writing out a batch of text code quite different in structure from the graphical representation, it will be the actual code presenting itself graphically that will appear in the editor. Students will be able to pick “if,” “else,” “case,” “+” and other language elements off a bean palette and connect them graphically. A bean will know what other types of beans and how many it can be connected to, and it can offer students help in creating a statement. For instance, if a student picks the “+” bean, the editor could indicate that all its input connections must evaluate to numbers. The student could attach numbers, symbols or functions to the bean only as input. (Unfortunately, we couldn’t tell in advance if a particular symbol or function would result in a number – to do that we’d have to be able to run the program before the student finished constructing it!)

HotScheme is an open source project, and we encourage sites interested in using the tool and developers who wish to contribute to contact us. The Web site for the project, which includes two applet versions, JavaDocs, sample Scheme code and an archive of the source code is http://stgtech.com/HotScheme/.

CODE LISTING

Code listings for this article can also be located at www.JavaDevelopersJournal.com

gcallah@erols.com bclark@crosslink.net

Listing 1

```
abstract public class LispTerminal extends
HotSchemeInternalRep
{
    public LispTerminal()
 {
 }
    publicLispTerminal(LispTerminal in,
    LispTerminal out)
 {
     in_term  = in;
     out_term = out;
 }
    public final void print(Object obj)
    {
    if(out_term != null)
         out_term.print(obj);
         else
    my_print(obj);
    }
    abstract public void my_print(Object
    obj);

    public final void print(int i)    {
      print(new String((new
      Integer(i)).toString())); }
    public final void print(long l)     {
      print(new String((new
      Long(l)).toString())); }
    public final void println(Object obj)
    {
         if(out_term != null)
         out_term.println(obj);
         else      my_println(obj);
    }
    abstract public void my_println(Object
    obj);
    public int read()
    {
         if(in_term != null) return
          in_term.read();
         else        return my_read();
    }
    abstract public int my_read();
    public void unread(int c)
    {
         if(in_term != null)
          in_term.unread(c);
         else          my_unread(c);
    }
    abstract public void my_unread(int c);
    public boolean eof()
    {
         if(in_term != null) return
          in_term.eof();
         else            return my_eof();
    }
    abstract public boolean my_eof();
    public SchemeToken getToken()
    throws SchemeException
    {
         if(token_stack.empty())
          return new SchemeToken(this);
         else      return
          (SchemeToken)token_stack.pop();
    }
    public void pushToken(SchemeToken
    token)
    {
         token_stack.push(token);
    }
//place for clients to push tokens back to:
  private Stack token_stack = new Stack();
    private LispTerminal in_term  = null;
    private LispTerminal out_term = null;
}
```

```
import java.io.*;
import java.util.Stack;
```

Listing 2

```
public class CharLispTerminal extends
LispTerminal
{
    public CharLispTerminal(InputStream
     in, PrintStream out)
    {
    input  = new PushbackInputStream(in);
    output = out;
    }
    public void my_print(Object obj)
    {
    output.print(obj); output.flush();
    }
    public void my_println(Object obj)
    {
    output.println(obj);
    }
    public int my_read()
    {
    int c = 0;
    try
    {
    c = input.read();
    }
    catch(java.io.IOException e)
    {
             ;
        }
        return c;
    }
    public void my_unread(int c)
    {
        try
        {
            input.unread(c);
        }
        catch(java.io.IOException e)
        {
            ;
        }
    }
    public boolean my_eof()
    {
    return false;
    }
    private PushbackInputStream input;
    private PrintStream output;
}
import java.util.Vector;
```

Listing 3

```
class GUILispTerminal extends
BufferedLispTerminal
{
 private Vector  m_vOutput;
 private boolean debug = false;
    public GUILispTerminal()
    {
  m_vOutput = new Vector(5);
 }
 private void print(String s)
 {
  m_vOutput.addElement(s);
  if(debug)
   {
System.out.println("GUILispTerminal::print
Vector is " + m_vOutput);
System.out.println("GUILispTerminal::print
" + s);
   }
 }
 public void setInput(String s)
 {
 setBuffer(s);
  if(debug) System.out.println("GUILispTer-
  minal::setInput " + s); //debug
 }
 public Vector getOutput()
 {
 if(debug) System.out.println("GUILispTer-
 minal::getOutput " + m_vOutput); //debug

  Vector vTemp = (Vector)m_vOutput.clone();
  m_vOutput.setSize(0);
  return(vTemp);
 }
    public void my_print(Object obj)
    {
  print(obj.toString());
    }
    public void my_println(Object obj)
    {
  print(obj.toString() + "\n");
 }
}
```

Listing 4

```
abstract public class BufferedLispTerminal
extends LispTerminal
{
    public BufferedLispTerminal()
 {
     super();
         CommandBuf = new StringBuffer("");
 }
    public BufferedLispTermi-
nal in, LispTerminal out)
 {
     super(in, out);
         CommandBuf = new StringBuffer("");
 }
    public int my_read()
    {
  if(pos < CommandBuf.length())
   return((int)CommandBuf.charAt(pos++));
  else
      return(0);
    }
    public void my_unread(int c)
    {
  if(pos > 0) pos--;
    }
    public boolean my_eof()
    {
         if(pos < CommandBuf.length())
         {
// if everything else is spaces, return
true, else false
             for(int i = pos; i < Command-
Buf.length(); i++)
             {

if(!Character.isWhitespace(CommandBuf.charA
t(i)))
                     return false;
             }
             CommandBuf.setLength(pos);  //
we won't look at those spaces again
         }
         return true;
    }
    protected void setBuffer(String s)
    {
         pos = 0;
         CommandBuf.setLength(0);
         CommandBuf.append(s);
    }
    private StringBuffer CommandBuf;
   private int pos = 0;
}
```

COMPLETE CODE LISTING INCLUDED!

Writing Effective Web Applications Using Java and Oracle

WRITTEN BY
VIVEK SHARMA

As the Web grows, a great deal of effort is being made toward writing applications in Java that interact with databases. Fortunately, JDBC provides an easy, database-vendor–independent way of writing such applications. While this approach works for a number of applications, there are limitations. For instance, if the application requires execution of a large number of SQL statements, efficiency becomes an issue since round-trips from the application to the database are costly. Also, since these statements aren't precompiled, the database system spends more time executing them than it would have if they were compiled and residing in the database.

This is where PL/SQL comes in. It's a database language that enhances SQL by providing structure to SQL programs. It's rich with elements like control structures that we're used to seeing in such languages as C and Java. Thus you can write arbitrarily complex routines that manipulate the database using PL/SQL. More important, you can compile PL/SQL programs in an Oracle database so they execute faster at runtime.

Oracle Application Server

The next question is, How do we execute PL/SQL programs from within our Java applications? Enter the world of the Oracle Application Server (OAS)! Web applications started with CGI programs running at the back end. With increased use, this approach started reaching its limits because most CGI programs consume a lot of resources on the host machine for execution. Application servers came on the market to solve this problem, among others. The Oracle Application Server is one such product that helps to execute your Web applications efficiently .

Since describing the OAS in detail isn't feasible here, I'll cover the basic items you'll need to understand the rest of the material. Broadly speaking, the OAS consists of a Web Request Broker (WRB) that handles Web requests that require execution of an application. The WRB routes these requests to the appropriate executable code. It provides load balancing and other services, such as transaction management, to the applications executing in the OAS environment.

Cartridges

The requests are served by programs called cartridges that in turn may execute applications written in languages such as Java, C and Perl. As a developer, you may write either a cartridge or a cartridge application, commonly referred to as a *component.*

The OAS comes with a number of cartridges, one of which is the JWeb cartridge. This cartridge is meant for executing Java applications. It maintains an instance of a Java Virtual Machine and executes the Java class specified in the request.

The OAS also comes with a PL/SQL cartridge, which can be used for executing PL/SQL applications directly. However, the focus of our current discussion is on integrating Java applications with database applications, so we won't discuss the PL/SQL cartridge here.

Virtual Paths

There is the concept of virtual paths in the OAS. A virtual path can be thought of as an arbitrary name that you specify through the administration screens of your OAS. Each virtual path maps to a particular cartridge. When a request comes in, the OAS checks to see if the URL contains a virtual path in it. If it does, the OAS determines which cartridge this path maps to and then directs an available instance of that cartridge to execute the code requested in the URL.

A virtual path also has a physical directory associated with it – this is where the executable code you've written needs to reside. The following example should make this clear.

Let's say you've declared a virtual path called "myjava" in the OAS that maps to the JWeb cartridge, and specified that the associated directory is /home/me/java_dir. If somebody sitting on a client such as a browser requests the URL http://yourmachine/myjava/Employees, the OAS will recognize myjava as a virtual path. It'll also determine that this maps to a JWeb cartridge. Accordingly, it'll direct an available instance of the JWeb cartridge to execute Employees.class. This is a class file created by you that resides in the directory /home/me/java_dir on the server where your OAS is running.

pl2java Utility

A utility called pl2java comes as part of the OAS package. If you invoke this utility on a PL/SQL package stored in your Oracle database, it produces a Java class file. You may then use this class file in your Java application to invoke procedures and functions in that package. (A PL/SQL package is a collection of PL/SQL procedures and functions grouped together.)

With this background about the OAS, we can move on to see how a program running in the JWeb cartridge can execute PL/SQL procedures and functions. To illustrate a typical usage, we'll develop a small application – one that allows people to register online on your Web site.

Developing a Sample Application

The registration process starts with a user accessing your registration URL. To keep things simple, let's assume this URL is a static HTML page. This page contains an HTML form in which the user fills out information such as first and last name, e-mail address, phone and so on. The "ACTION" of this form is an application that writes the information to an Oracle database and returns a thank you page.

Internally, this application is written in Java where variables from your form are extracted and error checking is done. A PL/SQL procedure that writes the information to the database and returns a success or failure code to the Java application is then invoked. Based on this result, the Java application returns a thank you or error page to the user.

Designing the Database

The first step in writing this application is to design the database where information (name, address, phone, e-mail address) will be stored. While it's quite reasonable to have just one table in which all the information will be stored, we'd be compromising extensibility of the database in doing so. Instead, let's see which things go together and which don't.

Obviously, the first and last name go together. So let's create a table called "Person" with two fields, First_Name and Last_Name, both of VARCHAR2 with max size of 50 each. Address can be in a separate table of the same name. Fields can be "Line_1", "Line_2", "City", "State", "Country" and "Zip". How about e-mail and phone? They're essentially telecom addresses. Keeping them in the same table will have the added advantage that if we want to expand the registration application later to include fax, mobile phone, etc., we won't have to write an additional table to store this new information. But we need to distinguish one type from another. So we'll have a "Type" field in this table that can take on values like "E-MAIL", "FAX" and "PHONE". This table would be "Telecom_Address" and would have the following fields: "Code" and "Type", both of which are VARCHARs.

A quick recap – we have three tables:

```
Person – First_Name, Last_Name
Address – Line_1, Line_2, City, State,
Zip, Country.
Telecom_Address – Code, Type.
```

We also need to establish a relationship between these tables so we can link the records. In the Person table we add a field called "Id". This will have a unique value for each person who uses the registration system. In the Address and Telecom_Address tables we add a field called "Person_Id" that points to a record in the Person table.

To illustrate how it'll work, assume that five people have registered. For the first user our application stores "1" in the ID field of record created for this user in the Person table, "2" for the second one and so on. It also stores the same number in the Person_Id field of records created in the Address and Telecom_Address table. To find the name, address and e-mail of the second person, we can issue the following SQL statement:

```
SELECT Person.First_Name,
Person.Last_Name,
       Address.Line_1, Address.Line_2,
Address.City,
       Telecom_Address.code
>FROM   Person, Address, Telecom_Address
WHERE  Person.Id = 2 AND Person.Id =
Address.Person_Id
       AND Person.Id =
Telecom_Address.Person_Id
       AND Telecom_Address.type =
'EMAIL';
```

PL/SQL for Manipulating the Database

Next, we'll write some PL/SQL packages that will insert records into the database. But as we saw, we need a way to generate a unique number for the ID field of Person. For this we'll use the concept of sequences. For our purposes let's create a sequence of the name "Person_Seq" by issuing the following SQL statement:

```
CREATE SEQUENCE Person_Seq START WITH 1;
```

As far as packages are concerned, we'll create four packages – one for handling insertion into each of the three tables, and a top-level package that calls routines in the other packages. We do this so only one API will be consistently exposed to the Java application that needs to call these routines.

I won't go into much detail about packages and PL/SQL in general – suffice it to say that a package consists of a declaration followed by an implementation, also known as the package body. A sample package for adding stuff to the Person table would look like:

```
CREATE PACKAGE Person_Pkg AS
  PROCEDURE add(I_FNAME  IN VARCHAR2,
                I_LNAME  IN VARCHAR2,
                I_ID     IN NUMBER);
```

Listing 1

```
  add(F_NAME IN VARCHAR2,
      ....  -- Other things like l_name, address etc.
      PHONE   IN VARCHAR2,
      RESULT_VAL OUT NUMBER) IS
      Temp_Result NUMBER;
      Temp_Seq     NUMBER;
  BEGIN
      /* Temp_Result is a temporary variable that
         indicates to the calling Java application
         whether the user information was recorded
         successfully or not. 1 indicates success */
      Temp_Result := 1;

      /* Create a unique Id for this user */
      SELECT Person_Seq.NEXTVAL FROM Dual INTO Temp_Seq;

      /* We insert name of the person along with the ID
         generated above by calling the add() procedure
         in the Person_Pkg package created above */
      Person_Pkg.add(F_NAME, L_NAME, ID);

      /* Similarly the address is added. The ID this
         time goes into the Person_Id field of the
         Address table */
      Address_Pkg.add(LINE_1, ...... ID);

      /* We need to add the phone and e-mail as 2
         separate records */
      Telecom_Address_Pkg.add(PHONE, 'PHONE', ID);
      Telecom_Address_Pkg.add(EMAIL, 'EMAIL', ID);

      /* If an exception occurs, result is set to
         0 so the Java application can treat it
         accordingly. Note that you can add similar
         exception handlers in the other packages
         also to have better control over generation
         of error messages.*/
      EXCEPTION
          WHEN OTHERS THEN
             Temp_Result := 0;
  END add;
```

Listing 2

```
/*The OAS comes with a number of packages that allow you to
decode the FORM variables as well as generate HTML. The “rdbms”
package allows you to establish a Session with the database for
invoking PL/SQL procedures/functions. */

import oracle.rdbms.*;
import oracle.plsql.*;
import oracle.html.*;
public class Register
{

/*We’re declaring a variable ‘s’ of the type Session – here Ses-
sion is a class that comes in the rdbms package. It represents
the connection we’re establishing with the database. We’re
declaring this as a global ‘static’ variable so that the same
Session can be used by multiple requests,
increasing program efficiency. */

     static Session s = null;
     public static void main(String[] args)
     {
          Register r = new Register();
          r.start();
     }
     public void start()
     {

/* Declare a variable of HTTP.class - a class that comes as part
of the OAS and can be used within Java applications to get infor-
mation about the current HTTP request such as FORM variables */

     HTTP http;
     ...
     String fName = http.getURLParameter("fName);

/*Similarly, get all other parameters such as lName, etc. Check
if values are valid by calling a method value sAreValid (not
shown). You could check things like whether the e-mail entered by
the user has a missing '@' and other things like this. */

          if(valuesAreValid(fName, lName ........))
          {
             try{
                writeToDB(fName, lName ......);
             }catch(Exception e){}
          }
          else
          {
                 errorHandler();
```

```
END Person_Pkg;
CREATE PACKAGE BODY Person_Pkg AS
  PROCEDURE add(I_FNAME IN VARCHAR2,
               I_LNAME IN VARCHAR2,
               I_ID    IN NUMBER)
  IS
  BEGIN
     INSERT INTO Person(FNAME, LNAME,
ID)
     VALUES (I_FNAME, I_LNAME, I_ID);
  END add;
END;
```

Packages for manipulating the other two tables would look similar.

Our top-level package is Reg_Api. The body of this package contains the procedure add(), which looks like Listing 1.

The next step is to compile these packages by using something like SQL*Plus. Once these packages are compiled, we can use the pl2java utility to generate a class file.

Invoking pl2java

If we've compiled these packages in a database in the schema "myschema/myschema", with the connect string as "myConnect", we can invoke the pl2java utility from the command line as follows:

```
pl2java myschema/myschema@myConnect
Reg_Api
```

This will generate a class called Reg_Api.class that can be used in our Java application.

Note that pl2java accepts some command-line parameters that enable you to do things like changing the name of the output class file.

HTML Page for the Application

Now that we know where and how we're going to store information, we'll take a top-down approach, starting with the HTML form. Instead of showing the complete HTML, I'll show some relevant portions:

```
<FORM ACTION='http://yourserver/myja-
va/Register'>
<INPUT TYPE='text' NAME='fName'
VALUE=''>
<INPUT TYPE='text' NAME='lName'
VALUE=''>
<P>
<INPUT TYPE='text' NAME='line1'
VALUE=''>
..... Other fields
<INPUT TYPE='text' NAME='email'
VALUE=''>
<INPUT TYPE='text' NAME='phone'
VALUE=''>
<P>
<INPUT TYPE='submit' NAME='Submit'
VALUE='Submit'>
</FORM>
```

This HTML file can be put in the document root directory of the machine hosting the OAS. We now need to create a Java application – specifically, a class called Register. After we've created it, we need to move it to the directory that corresponds to the virtual path "myjava".

The Java Application

Our Java application should first decode the values typed by the user and check to make sure they're valid. Then it should invoke the PL/SQL procedure developed above. Based on the result value returned by the PL/SQL procedure, it should commit the transaction and present a thank you page, or roll back and present an appropriate error page.

Listing 2 shows what our Java program looks like.

Conclusion

As we've seen, PL/SQL procedures can be invoked easily through an OAS environment. This saves us unnecessary round-trips to the database, and gives us an added advantage in that the SQL code is precompiled and would thus execute faster. All in all, by using the pl2java utility of the OAS, you can create a seamless interface between your Java applications and PL/SQL procedures to build powerful Web applications that interact with the database.

(The opinions and statements expressed in this article are my own and don't necessarily represent those of Oracle Corporation.)

vivek_sharma_99@yahoo.com

AUTHOR BIO

Vivek Sharma is a software developer at Oracle Corporation. He has a BS in computer science and some six years of software research and development experience. Interests include development of Web-based tools/technologies and technical writing.

```
           }
     }
     void writeToDB(String fName, ......)
           throws ServerException
     {

// Before we create a Session, we need to inform it of the ORA-
CLE_HOME.

          Session.setProperty("ORACLE_HOME", OracleHome);

/* Check if there is a session already established with the data-
base. If so, use that Session else create a new Session */

          if(s == null)
               s = new Session("userName",
                           "password", "connectString");

/* As we can see here, if a session has already been established
in this instance, the same session can be used by other HTTP
requests handled by this instance. Since a good deal of time is
established in creating connections to the database, we have cre-
ated an optimization that will ensure the session need not be
reestablished for every new request. */

/* Reg_Api is the class generated by pl2java */

          Reg_Api ra;

/* Create an instance of Reg_Api and pass the instance of the
session created above so the class knows which database/schema it
needs to work against */

          ra = new Reg_Api(s);

/* Since there are no datatypes in Java that map directly to
PL/SQL datatypes, Oracle has provided mapping classes in the
plsql package. For a VARCHAR2, you need to convert your String to
a PStringBuffer() before calling a PL/SQL procedure that requires
a VARCHAR2 as input. Similarly for NUMBER, you must create a
PDouble. Likewise there are equivalent datatypes for most PL/SQL
datatypes. */

          PStringBuffer pFName = new PStringBuffer(fName);

// Similarly PStringBuffers for all variables to be passed to
add() in Reg_Api

          ...

/* The result, however, is a NUMBER type, so we create a PDouble
*/

          PDouble result = new PDouble(1);

/* Now invoke the procedure as if you were calling a normal
PL/SQL procedure */

          ra.add(pFName, pLName, ....pPhone, ... result);

/* result will now contain the value stored in this variable by
the add() procedure. We need to examine this. Before that we con-
vert it to an int by calling a method in PDouble */

          if(ra.intValue() == 0)
          {
               errorHandler();

               /* Rollback this transaction as it failed */
               s.rollback();
          }
          else
          {
              /* Commit this transaction */
              s.commit();
              showThankYouPage();
          }
     }
}
```

CODE LISTING

Code listings for this article can also be located at www.JavaDevelopersJournal.com

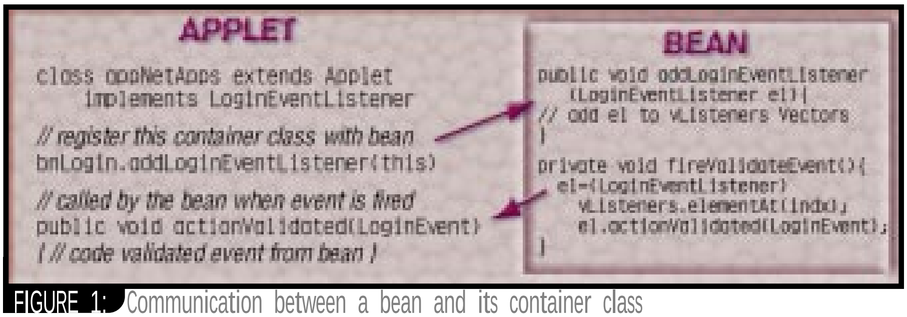
Adding a Custom Event to a JavaBean

Bidirectional communication between your custom bean and its container classes

COMPLETE CODE LISTING INCLUDED!

WRITTEN BY
JIM MANGIONE

One of the first truly reusable components I wrote in Java was a login bean that validated a username/password against our company's network. It was lightweight (using AWT classes), and worked with both applets and applications.



```
APPLET
class appNetApps extends Applet
    implements LoginEventListener
// register this container class with bean
bnLogin.addLoginEventListener(this)
// called by the bean when event is fired
public void actionValidated(LoginEvent)
{ // code validated event from bean }
```

```
BEAN
public void addLoginEventListener
   (LoginEventListener el){
// add el to vListeners Vectors
}
private void fireValidateEvent(){
  el=(LoginEventListener)
     vListeners.elementAt(indx);
     el.actionValidated(LoginEvent);
}
```

FIGURE 1: Communication between a bean and its container class

This proved to me how simple writing and deploying a JavaBean was. It also introduced me quickly to the concept of custom event handling. You see, the bean's container class (be it an applet, frame, etc.) can easily talk to the bean and invoke public methods such as bnLogin.setDefaultUserName(). But after the bean validates a user, it has to notify its container whether the user is valid or invalid through the use of a custom event.

In JDK 1.1, if you use only standard GUI events such as mouseClicked() or keyPressed(), they're inherited from the java.awt.AWTEvent, which superseded the java.awt.Event class used in JDK 1.0. If a TextField component wants to inform its container class that a user just typed a character into it, there's a standard event for that, keyPressed(), which most of today's IDEs will set up for you. However, if a custom component such as our login bean wants to inform its container class that a user has just been validated – actionValidated(), for instance – a new event must be written.

Along with the JDK 1.1 java.awt.AWTEvent class came the introduction of the java.util.EventListener and java.util.EventObject classes. These exist to provide custom event handling. To use them is a four-step process.

Step 1

In Listing 1 a new event is created by extending the EventObject class, which will hold the username and validated status when the bean fires a login event.

Step 2

A new listener is created as an interface that extends the EventListener class, which contains the methods that pass the LoginEvent object to all registered listeners, as shown below:

```
public interface LoginEventListener
extends EventListener {
   public void actionValidated(Login-
Event e);
   public void actionCanceled(Login-
Event e); }
```

The container class (such as an applet) will implement this listener and the two defined methods. The login bean will store a reference to the container class (stored in a vector) and, using that reference, call the appropriate method whenever an event is fired (see Figure 1).

Step 3

Next, our login bean will need to be modified to create LoginEvents and notify the container classes of an event via the methods defined in the LoginEventListener. It must also provide public methods to add and remove the container classes. Don't worry. I can assure you it's really not as bad as it sounds!

We'll start with how to register the container classes that will listen to events fired by the bean. As shown in Listing 2, an addLoginEventListener() and removeLoginEventListener() are added as public methods to the bean. These maintain the vListeners Vector, which holds LoginEventListener objects. There will be one element in this Vector for each listener.

The two events required by our login bean to be fired are when a user selects the <OK> button, actionValidated(), and if the user cancels the login, actionCanceled(). The easiest way to do this is to have two private methods called fireValidatedEvent() and fireCanceledEvent(). After the user has been validated or denied access, I call the callfireValidateEvent() method or the fireCanceledEvent() method inside the mouse-click event of the cancel button. Listing 3 shows an example of the fireValidatedEvent. This loops through each element in the vListeners Vector, casting the element to the LoginEventListener class and calling the actionValidated() method of that object.

The same code is used for the fireCanceledEvent(), only el.actionValidated() is replaced with el.actionCanceled().

Step 4

The login bean is now firing off events for any objects that implement the LoginEventListener class and register with the bean. So how do we listen for these events? The container class must implement the LoginEventListener class, register itself with the bean and define the methods from the listener. This is detailed in Listing 4.

By following these four steps, you can have bidirectional communication between your custom bean and the container classes that hold them.

AUTHOR BIO
Jim Mangione, a senior database application developer with an MS in computer science from Drexel University, has nine years of IT experience in the pharmaceutical and chemical industry. He's been involved in both traditional client/server development and distributed application development using Java and application server technology.

mangione@k2nesoft.com

Listing 1

```
public class LoginEvent extends EventObject {
   private String sUSERNAME;
   private boolean bVALIDATED;

   LoginEvent(Object source) {
      super(source);
   }
   LoginEvent(Object source, String sUsr, boolean bValidated)
{
         this(source);
         sUSERNAME = sUsr;
         bVALIDATED = bValidated;
   }
   public String getUsername() { return sUSERNAME; }
   public boolean getValidated() { return bVALIDATED; }
}
```

Listing 2

```
public addLoginEventListener( LoginEventListener el ) {
   vListeners.addElement( el );
}
public removeLoginEventListener ( LoginEventListener el ) {
   vListeners.removeElement( el );
}
```

Listing 3

```
private void fireValidatedEvent(String sUsr,boolean bValidat-
ed) {
   Vector v;
   LoginEvent e;
   v = (Vector) vListeners.clone();
   e = new LoginEvent(this, sUsr, bValidated);
   for (int i=0; i<v.size(); i++) {
   LoginEventListener el = (LoginEventListener)
      v.elementAt(i);
      el.actionValidated(e); }
}
```

Listing 4

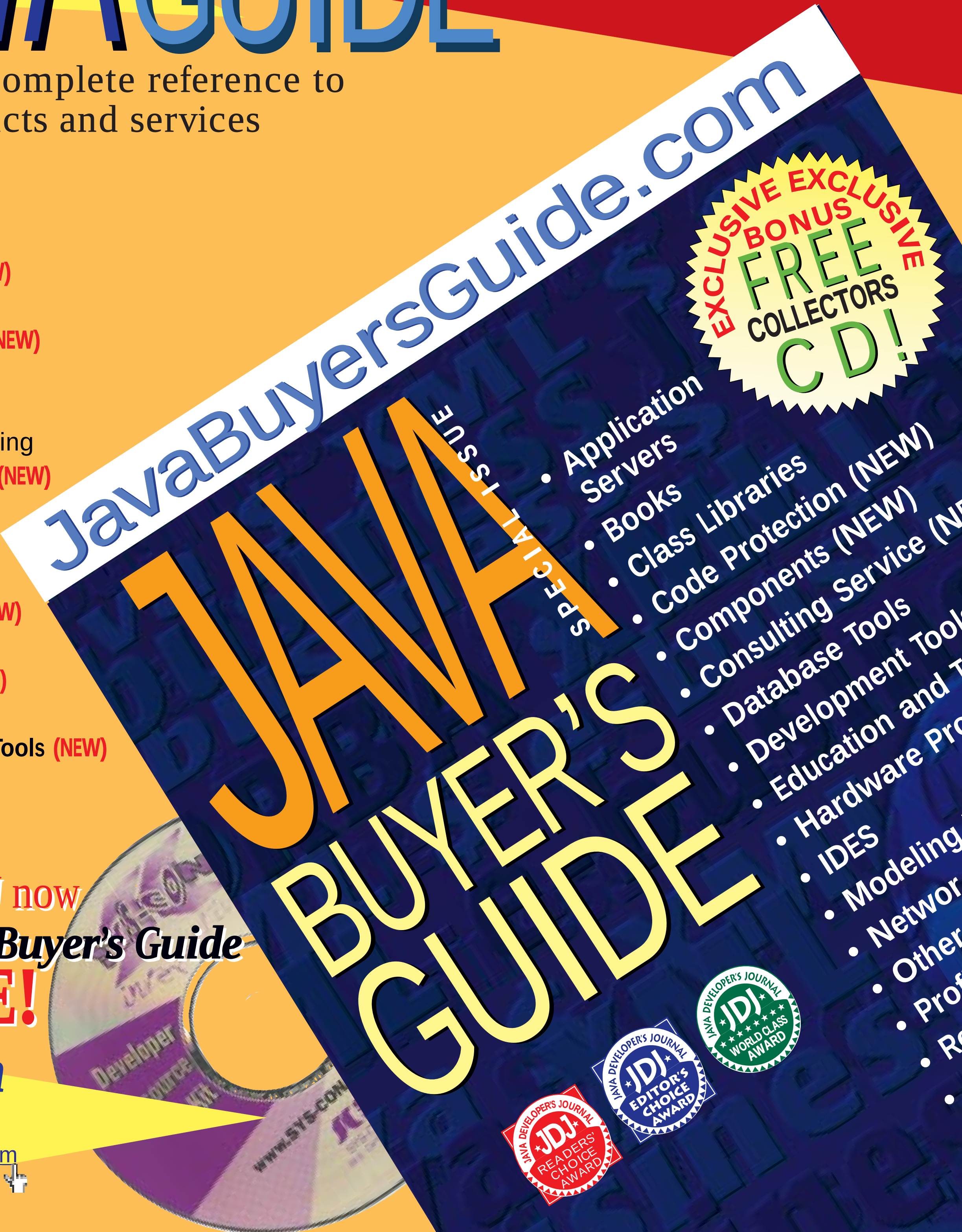
```
public class appNetApps extends Applet implements
LoginEventListener {
***
***
bnLoginBean bnLogin = new bnLoginBean();
bnLogin.addLoginEventListener( this );
// adds this object to the vListeners Vector in the bean
***
***
public void actionValidated(LoginEvent e) {
// code here for handling event when user clicks <OK> on
// login bean
}
public void actionCanceled(LoginEvent e) {
// code here for handling event when user clicks <Cancel> on
// login bean
}
```

▼▼▼▼▼▼ **CODE LISTING** ▼▼▼▼▼▼

The code listings for this article can also be located at
www.JavaDevelopersJournal.com

ADVERTISING INDEX

ADVERTISER	URL	PH	PG
9NETAVENUE, INC.	WWW.9NETAVE.COM	888.9NETAVE	49
BATEMAN, INC.	WWW.BATEMANINC.COM	805.383.3338	75
BEA WEBLOGIC	WWW.WEBLOGIC.BEASYS.COM	800.817.4BEA	2
BLUE SKY SOFTWARE	WWW.BLUE-SKY.COM	800.559.4423	13
BORLAND.COM	WWW.INTERBASE.COM/PRODUCTS/DEMOJDJ.HTML	800.451.7788 x7183	21
CAREER OPPORTUNITY ADVERTISERS		800.846.7591	76-81
CEREBELLUM SOFTWARE	WWW.CEREBELLUMSOFT.COM	888.862.9898	23
CLOUDSCAPE, INC.	WWW.CLOUDSCAPE.COM	888.59JAVA1	15
COMPUTER ASSOCIATES INTERNATIONAL, INC.	WWW.CAI.COM/ADS/JASMINE/DEV	888.7JASMINE	6
CYRUS INTERSOFT, INC.	WWW.CYRUSINTERSOFT.COM	612.331.6600	39
CYSCAPE	WWW.CYSCAPE.COM/FREE4J	800.932.6869	16
DEVELOPMENTOR	WWW.DEVELOP.COM	800.699.1932	63
ELIXIR TECHNOLOGY	WWW.ELIXIR.COM.SG	65 532.4300	35
ENTERPRISESOFT	WWW.ENTERPRISESOFT.COM	510.742.6700	11
FINDAHOST.COM	WWW.FINDAHOST.COM	440.257.6690	69
HOSTPRO	WWW.HOSTPRO.NET	213.252.9779	57
IMI SYSTEMS INC.	WWW.IMISYS.COM	800.828.0180	61
INETSOFT TECHNOLOGY CORP	WWW.INETSOFTCORP.COM	732.235.0137	47
INSIGNIA SOLUTIONS, INC.	WWW.INSIGNIA.COM	800.848.7677	33
INSTANTIATIONS INC.	WWW.INSTANTIATIONS.COM	800.808.3737	67
JAVA DEVELOPER'S JOURNAL	SUBSCRIBE@SYS-CON.COM	800.513.7111	73-75
JDJ STORE	WWW.JDJSTORE.COM	888.303.JAVA	59
KL GROUP INC.	WWW.KLGROUP.COM/JCLASS/POWER	888.328.9597	52,53
KL GROUP INC.	WWW.KLGROUP.COM/CULPRITS	888.328.9597	84
KL GROUP INC.	WWW.KLGROUP.COM/TRUTH	888.328.9597	25
NSICOM	WWW.NSICOM.COM	97.2.353.31976	63
OBJECT DOMAIN SYSTEMS, INC.	WWW.OBJECTDOMAIN.COM	919.461.4904	37
OBJECT INTERNATIONAL SOFTWARE	WWW.TOGETHERJ.COM	919.772.9350	41
OBJECTSPACE, INC.	WWW.OBJECTSPACE.COM/GO/UNIVERSAL	800.OBJECT1	82,83
OBJCETSPACE, INC.	WWW.OBJECTSPACE.COM	888.59JAVA1	45
ONEREALM, INC.	WWW.ONEREALM.COM/JDJ	800.633.1072 x23839	7
ORACLE CORPORATION	WWW.ORACLE.COM/INFO/32	800.633.1072 x23839	17
PROTOVIEW	WWW.PROTOVIEW.COM	800.231.8588	3
RIVERTON SOFTWARE CORPORATION	WWW.RIVERTON.COM	781.229.0070	29
SALES VISION	WWW.SALESVISION.COM	800.275.4314	51
SLANGSOFT	WWW.SLANGSOFT.COM	972.375.18127	55
SOFTWIRED INC.	WWW.SOFTWIRED.COM	(41)1.445.23101	38
SPECIALIZED SOFTWARE	WWW.SPECIALIZEDSOFTWARE.COM/JDJ/	800.328.2825 x6576	44
SUMMER INTERNET WORLD 99	WWW.INTERNET.COM/REGISTERSUMMER	800.632.5537	65
SUN MICROSYSTEMS, INC.	WWW.SUN.COM/SERVICE/SUNED	800.422.8020	4
THE THEORY CENTER	WWW.THEORYCENTER.COM	888.843.6791	43
TIDESTONE TECHNOLOGIES, INC.	WWW.TIDESTONE.COM	888.880.0665	31
TRAINING/ETC.COM	WWW.TRAININGETC.COM	410.531.9953	69
VISUALIZE INC.	WWW.VISUALIZEINC.COM	602.861.0999	63


JavaBuyersGuide.com
EXCLUSIVE EXCLUSIVE
BONUS
FREE
COLLECTORS
CD!
SPECIAL ISSUE
JAVA
BUYER'S
GUIDE
• Application Servers
• Books
• Class Libraries
• Code Protection (NEW)
• Components (NEW)
• Consulting Service (N
• Database Tools
• Development Tool
• Education and T
• Hardware Pro
• IDES
• Modeling
• Networ
• Other
• Prof
• R
JAVA DEVELOPER'S JOURNAL JDJ READERS' CHOICE AWARD
JAVA DEVELOPER'S JOURNAL JDJ EDITOR'S CHOICE AWARD
JAVA DEVELOPER'S JOURNAL JDJ WORLD CLASS AWARD
Developer
www.SYS-CON

(continued from page 7)

Another view was that XML would replace proprietary business-to-business commerce systems based on EDI, and that it would open up new models of e-commerce based on industry-standard XML schemas. This view, sometimes referred to as the XML data worldview, has increasingly become the more relevant model for XML usage.

Despite an emerging clarity around the "proper" use of XML, adoption has remained scattered. A first barrier to adoption was that the XML standards weren't solid. This has changed for the better in the last six months. A second major issue was – and is – the fact that most data and application environments haven't been set up to use XML. Related to this has been the fact that few developers are conversant with this new model for data and application integration.

The Emerging Distributed Web

About a year ago I wrote a series of white papers under the banner of the "emerging distributed Web." The basic argument was pretty simple. The Internet was about to undergo a radical transformation based on new business models centered around Web syndication. Syndication represented the idea that every Web site was *not* an island, that the content and application assets of every Web system was a set of data and services that could be leveraged by every other site on the network.

Prior to writing the papers, I had been influenced by Bob Metcalfe, who anticipated this in what he dubbed "Metcalfe's Law," which stated that for every *n*-node added to the Internet, the Internet's value would grow exponentially. In this law I saw that if every Web system was a set of data and application APIs for every other Web system, this would radically change how business would be built in the emerging Internet era.

From this it became clear that there needed to be an extremely simple model for Web applications and systems to share assets with each other. Of course, XML had recently emerged as the proper foundation for this, but the basic XML infrastructure would not be sufficient to meet the needs of developers and companies wanting to do it easily.

What was needed was a pragmatic view of distributed Web computing, one that embraced the basic principles that have made the Web successful to date – principles such as simplicity and openness, and heterogeneous platforms and languages…and the idea that implementing this should not require an extremely advanced programming and systems infrastructure.

XML as Middleware

The basic conclusion drawn from all of this was that to embrace the opportunity of Web syndication, we needed to enable transparent application and data exchange based on XML but without all of the overhead of "purist XML." In short, we needed a model based on XML as middleware. This refers to using XML as a "tunnel" for application data exchange, where the applications and language environments that are communicating never know they are using XML as a transport. Internet programming languages would be the starting point, and a lowest-common-denominator approach would be developed to enable common data structures to be shared transparently between languages, without any real work by developers.

From this emerged the Web Distributed Data Exchange, or WDDX. Initially created by Allaire, and integrated into the ColdFusion language environment, WDDX was released into the public domain under a modified BSD license with an SDK and free implementations available from Wddx.org. Over the past six-months WDDX has been implemented to support COM (ASP, VB, etc.), Java, Perl, PHP, Python, JavaScript and ColdFusion.

All of a sudden, with just a Web server, a scripting language and HTTP, any application implemented in any language on the Internet can communicate with any other system. Content and application syndication would now be possible with a trivial amount of work by either party without their having to know a thing about XML.

The Loosely Coupled Web

The emerging XML middleware model, in fact, is posing a challenge to those who are investing entirely in binary distributed object standards. Indeed, Web systems will increasingly be implemented for maximum interoperability with other applications on the Internet. Given this, a model based on XML, which relies on loosely coupled systems communicating over HTTP, and implemented in potentially any language environment, is changing architectural decisions for many Internet-centric companies.

By building on a simple foundation such as WDDX, one can imagine layering additional semantics and services that support standard RPC-style invocations, or more robust object request brokers layered on WDDX, but without any language bindings or dependencies, and potentially implemented over asynchronous transports such as SMTP and POP. Some analysts are even beginning to argue that XML-HTTP will evolve into a Web Object Model that rivals and even replaces systems such as CORBA or EJB.

Designing for Web Syndication

Whatever evolution this new platform model actually takes, it is clear that systems built around Web syndication represent an enormous opportunity for companies and developers. Embracing this new model requires a few simple things.

- First, companies must determine what business model they want to wrap around Web syndication. For example, an e-commerce site might create an affiliate program, and enable site affiliates to remotely grab content and invoke transactions from your site through an XML-based API. Or, in a B2B model, a company could expose a lead-tracking system through a private URL, syndicating lead data to partners accessing the system from their own applications.
- Second, companies need to evolve their core Web application infrastructure to support Web syndication more directly. A great start for this would be a JSP server, such as Allaire JRun, and WDDX for Java, available from Wddx.org. From this it becomes trivial to expose any data or application services to other applications, and to consume content and application data from other sites.
- Finally, companies need to determine what content and services they want to expose, build special URLs for accessing those functions and document this for their customers and partners. I like to think of this as developing a Web SynDK for your business – a set of Web-based APIs, exposed through XML or WDDX, that enable your partners to integrate your business with theirs.

Cyrus Intersoft

Speiros delivers true anytime, anywhere pervasive computing

WRITTEN BY SCOTT DAVISON

AUTHOR BIO

Scot Davison is one of the founding authors of SYS-CON Publicaitons, Inc.., the publisher of Java Developer's Journal. Scott can be reached at scott@sys-con.com

scott@sys-con.com

Three years ago, Cyrus InterSoft's founder and CEO, Scott Bayless and his team of developers decided to make Java deliver on its promise of platform independent network computing. They recognized the need for a simple, cohesive way to transparently tie together heterogeneous networks and communications systems, providing users with computing access anywhere, anytime, on any device.

To fulfill Scott's vision, certain parameters would have to be met. The system would have to:

- Be entirely abstracted in order to sit on top of and leverage existing networks and operating systems without compromising security.
- Provide an entirely new level of access so users can be on any machine at any time.
- Allow for comprehensive application and resource allocation and distribution across systems and platforms.
- Contain a new economic model to enable effective distribution of those applications and resources.

After three years of R&D, Speiros was born.

On-Demand

Speiros provides an on-demand Java application launching platform, eliminating the barriers between the software vendor and the consumer. Users can now invoke and execute Java applications locally on their machines without having to install them. The Java application or applet does not require a browser and can reside on any server anywhere on the Internet. "Java applications and applets do not have to be 'Speiros aware' and therefore can be distributed and accessed on-demand without any installation or integration," says Bayless. "It's that easy. Now the user doesn't need to worry about installations or upgrades and manufacturers don't need to shrink-wrap and distribute products."

The old model of Web pervasiveness tied enterprises together through shared access to content and messaging services such as e-mail and FTP. Speiros adds a new layer to the enterprise, tying networks together through shared access to programs, files and other computing resources.

Single VM

Cyrus InterSoft made a number of technological breakthroughs while developing Speiros. One example is the ability to run multiple Java programs on a single VM without requiring any modifications to the VM. In addition, Speiros ensures that applications' run states are encapsulated so that no application can attack another.

NO APIs

A second breakthrough was developing a Java platform that required no APIs. Developers simply have to write clean Java code and it will run on the Speiros system.

Client-Side Java

Current invocations of Java technology tend to run on the server side. Cyrus Intersoft designed Speiros to run the Java programs on the client side. To make this possible, Cyrus Intersoft built into the system the ability to find, access and run – without manual installation – Java located anywhere on the Internet.

Beyond the Browser

Speiros was designed to go beyond the browser – users download the Java front end called an ESTATE. It provides a means to launch Java applications and applets as well as a networked pervasive file system.

The back end is made up of four servers: speirosCenter, resourceCenter, appCenter and meterCenter. Together these centers make it possible for the user to store configurations online, and find and rent applications and other computing resources on demand.

Capabilities

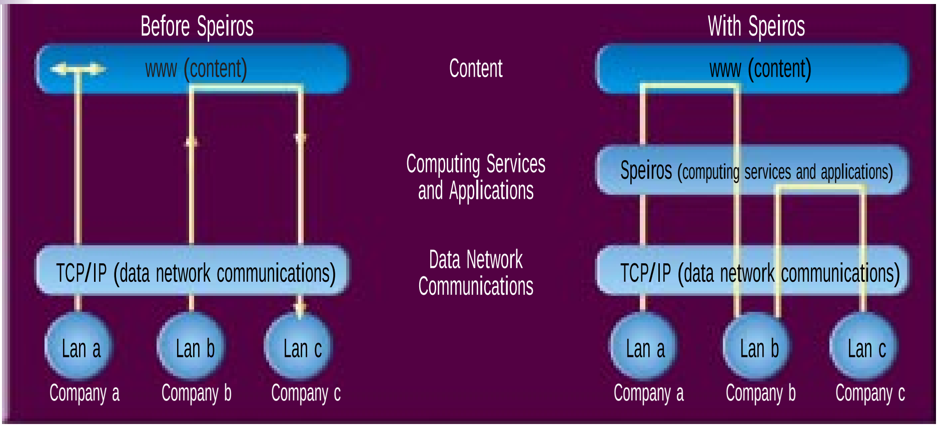
Speiros provides Java developers with the following capabilities:

- Internet access to their Java application or applet without any FTP or installation hassles
- Exchange or delivery of documents without attaching them to e-mail
- Instant upgrades of all user software with a single centralized copy
- Use on a rental or subscription basis

The Speiros platform enables developers and service providers to deploy their services in a secure environment that is available anytime, anywhere, from any network-enabled device.

Free Download

Developers can download a limited complimentary version of the Speiros technology in early August at www.cyrusintersoft.com.


Before Speiros
www (content)
TCP/IP (data network communications)
Lan a
Lan b
Lan c
Company a
Company b
Company c
Content
Computing Services and Applications
Data Network Communications
With Speiros
www (content)
Speiros (computing services and applications)
TCP/IP (data network communications)
Lan a
Lan b
Lan c
Company a
Company b
Company c