EDITOR'S CHOICE AWARDS WINNERS ANNOUNCED pg.42

JAVA DEVELOPER'S JOURNAL

JavaDevelopersJournal.com

Volume: 3 Issue: 12, 1998

SYS-CON PUBLICATIONS

JAVA DEVELOPER'S JOURNAL

EDITORIAL ADVISORY BOARD

Ted Coombs, Bill Dunlap, David Gee, Michel Gerin, Arthur van Hoff, Brian Maso, John Olson, George Paolini, Kim Polese, Sean Rhody, Rick Ross, Richard Soley,

Editor-in-Chief: Sean Rhody
Art Director: Jim Morgan
Executive Editor: Scott Davison
Managing Editor: Hollis K. Osher
Senior Editor: M'lou Pinkham
Production Editor Brian Christensen
Technical Editor: Bahadir Karuv
Visual J++ Editor: Ed Zebrowski
Visual Café Pro Editor: Alan Williamson
Product Review Editor: Jim Mathis
Games & Graphics Editor: Eric Ries
Tips & Techniques Editor: Brian Maso

WRITERS IN THIS ISSUE

Tim Callahan, Lisa Chiranky, Joe Chou, Andrei Cioroianu, Daniel Dee, Claude Duguay, Diane Hagglund, Anil Hemrajani, George Kassabgi, Jim Milbery, David Norris, Snehal Parikh, Sean Rhody, Ajit Sagar, 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
Advertising Assistants: Robyn Forma, Jaclyn Redmond
Accounting: Ignacio Arellano
Graphic Designers: Robin Groves, Alex Botero
Webmaster: Robert Diamond
Customer Service: Sian O'Gorman, Paula Horowitz
Online Customer Service: Mitchell Low

EDITORIAL OFFICES

SYS-CON Publications, Inc.
39 E. Central Ave., Pearl River, NY 10965
Telephone: 914 735-1900 Fax: 914 735-3922
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

SYS-CON PUBLICATIONS

FROM THE EDITOR

Sean Rhody, Editor-in-Chief

Sudden Impact

Every now and then I like to step back from the trenches and try to think like a CIO. I was a CIO at one time, so I can actually do such a thing. And lately, when I think my CIO thoughts, I've been thinking about the impact that Java has made on the Enterprise.

That impact tends to be a great deal different than it would be to a programmer. To a programmer, the game is about having a language that makes it easy – and fun – to do the task at hand. You don't see hordes of programmers moving to COBOL to write Internet programs – it's not the right tool. It's not fun in COBOL. You pick a language that makes sense to write in.

At the high level though, what makes sense "for the company" may be in stark contrast to what makes sense at a micro level. There the decision may not involve what language to use, but whether it makes sense from a business standpoint to even be writing Internet programs, especially if your staff is mainly COBOL programmers.

When I examine Java from that dizzying height, I see an interesting facet. Java is not really aimed at winning programming wars. It's aimed at hardware. Let me tell you why. On the surface, and in the mind of the individual programmer, Java is primarily a software movement. An open software movement, one that says write once, run everywhere. The goal of Java is that any Java bytecode should run on any Java VM, from any vendor on any operating system running on top of any equipment. Further, Java is about removing the need for an operating system, at least in certain cases, such as with embedded processors.

So how is that about hardware? Here's how. Once we have a substantial base of code in Java, Java Applications, Java Applets, JavaBeans and EJBs, the investment will be made in software. At that point Java becomes one of the standards by which the Enterprise is run. We'll have application servers to house our code, they'll talk to each other using RMI or CORBA and we'll have a lot of software that can run in a lot of places. So far so good?

Now we get to the $64,000 question. What do we run it on? Let's forgo applets and applications for the moment and consider a business that has developed a set of partitioned applications. Java will run everywhere, so where do we run our business? The answer is…on the platform that offers the best combination of scalability, interoperability and affordability to our company. And that's mainly a hardware question. I'll grant you that you can run several operating systems on some hardware, but for the most part server hardware is so tied to the operating system that you can consider them a single unit. No one buys a SUN box to run Linux in a production environment. You can have any operating system you want on an AS/400 – as long as you want OS/400. Given that the business logic is unseen, there's no need to argue over who has the better GUI. The choice comes down to what hardware is best for the task. We can compare TPC benchmarks, MFLOPs or any other standard that's meaningful in order to determine what's best for our company. The hardware vendor that will win the game is the one that ultimately understands how to combine processing power, affordability, service and marketing in a way that's cost effective to its clients and profitable to itself.

I'll admit this still doesn't help with the desktop, but let's face it, the war's over there. Microsoft won. But it does offer alternatives when the main corporation application is developed in Java as opposed to being Microsoft Office. If a browser is all that most people need, we might be able to change the face of the desktop in the Enterprise. Of course, the users may take the "You can have my PC back when you pry the mouse out of my cold dead fingers" attitude, and that's likely to be a significant impediment to the NC concept. Even Larry Ellison has begun to recant on the feasibility of NCs in an era of cheap PCs. Still, the possibility exists for dramatically decreasing costs for the desktop by providing most of the corporate applications via a browser, or via Java applications.

So much for the CIO hat. I'll hang that up for a while and put on my construction hard hat, then get ready for some heavy-duty Java development. It's fun, and anything's better than COBOL. Thanks for reading, and have a great holiday season.

About the Author

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. He can be reached by e-mail at sean@sys-con.com.*

GUEST EDITORIAL

Snehal Parikh

Managing Web Applications:
Delivering on the Internet's Promise

The Internet has evolved into an electronic marketplace, and businesses are increasingly realizing its benefits and those of associated technologies -- Web servers, browsers and Java. The next step is to leverage Web technologies so as to deliver mission-critical applications to employees, supply chains and customers. The flexibility and open standards of the Internet are what make intranets and extranets powerful, competitive weapons. Companies are seeing a world in which applications can be accessed by devices requiring nothing more than a Web browser as an operating environment.

While this is all very intoxicating, managing Web applications in a production environment can be sobering. Running applications within browsers creates challenges that must be addressed. To realize ultimate benefits, companies must ensure secure access to Web applications, include Web clients in their overall enterprise system's management initiatives and recognize a true competitive advantage by delivering "smart" Web applications.

Adaptable Web solutions are vital. Successful companies deliver customized views of applications and data via browser based on the type of employee, business partner or customer attempting access. No longer is generic access acceptable; businesses are expected to respond to individuals in regard to their personal requirements. This introduces the "smart" Web application that can detect information about the user, device, connection type and overall runtime environment, and adapt itself accordingly.

Many companies are coding this personalization logic into their Web technologies (i.e., using scripting tools and languages), but while it works, developing and maintaining scripted personalized Web-based applications can be a nightmare. A logical solution is to deliver an infrastructure that manages the personalization across all Web applications by providing services to manage preferences at the specific user, group and ultimate device level and allowing applications to take advantage of these settings. Web application developers can then focus on coding business logic and let the underlying infrastructure transform applications into "smart applications."

Many companies equate putting applications on Web application servers with exposing their assets to anyone with a browser. They need to set policies so only authorized users can access their mission-critical Web-based applications. The challenge is twofold: delivering applications only to authorized users, and centrally managing policies across multiple Web deployments in a unified manner.

Increasingly, enterprises are deploying multiple Web solutions across lines of business and departments. Each new deployment introduces Web administrators to new or duplicated administration and management schemes, increasing their management challenges. Even companies that deploy the same Web solution throughout the enterprise face the challenge of managing replications of the same user definitions across multiple domains.

The solution is a centrally controlled, unified schema that defines users, groups and systems with a common repository for the information delivered, using Lightweight Directory Access Protocol (LDAP), which is an emerging industry standard for network directories. Centralized schemas and repositories allow administrators to define policies once, for access by multiple tasks, which reduces their overall cost and complexity of management.

Web clients also introduce new challenges to administrators whose focus is to incorporate them into the overall enterprise system's management initiatives, which ensures reliability, availability and serviceability. Traditional system management tools monitor networks at the specific nodes where agents are deployed, but deploying agents at each node is cost prohibitive and impractical. The inherent flexibility of the Web allows users to be at their workstation or on a laptop, and ultimately on a pervasive computing device (such as PDA or smartphone).

A practical solution doesn't invent a new management scheme for Web clients, but incorporates them into existing management systems. The ability to generate notifications of alarms and events at the Web client and provide awareness of them with centralized system management consoles opens up the previously closed world of Web clients to administrators. The net result is a demystification of activities at the Web client by extending visibility and satisfying a pressing concern for administrators.

Who Delivers This Promise?

These issues are as new as the Web technologies they enhance. While many Web infrastructure vendors claim to develop, deploy and manage Web applications, they fall short at the enterprise level. They deploy and manage at the domain level, leaving the issue of unified management across the enterprise.

The recently introduced eNetwork On-Demand Server from IBM is a cross-platform server-based solution for the advanced deployment and management of Web-based clients. It improves Web-served computing by managing smart Web applications containing information about users, devices and connections -- allowing administrators to centrally manage client applications while reducing network computing and administrative support costs.

As more companies realize the challenges and benefits of Web technologies, new entrants will crowd the field, but IBM is already bringing its years of Enterprise Class understanding to the Web space.

About the Author

Snehal Parikh recently joined IBM as a product manager responsible for enterprise Web serving. He can be reached at sparikh@us.ibm.com.

PERSISTENT Threads for Friendly Applets

How many times have you downloaded an applet that couldn't be stopped?

by **Andrei Cioroianu**

Sometimes Java applets continue their execution even after the page that contains them is no longer visible. Run a few of them and your computer will slow down dramatically. If you continue you might need to reboot the system to avoid a crash. So you disconnect, reboot, reconnect and start all over again. Isn't it simpler to just disable the applets?

Yes. But you'll lose something if you do. Java isn't only for animations and cute navigation tools (though this article contains a lot about animation). Today everything moves on the Internet: businesses, services, entertaining and more. Suppose you need information today. If not today, then no later than tomorrow. You have three options:

1. Use the phone and speak with an operator, who will use a Java/native application running on a thin/fat client to query a database. A slow human intermediary will get the information, which will be communicated orally. You might have to note it on a piece of paper...
2. Connect directly to the company's Web site. Fill out a form and send the data to a servlet/CGI script that will return a page generated dynamically. If you don't get the requested information, you'll have to fill out the form again and repeat the procedure...
3. Or use a smart applet that guides you like a wizard and helps you get the information. You'll have a dynamic user interface, context-dependent help, friendly messages, tooltips, "Back" and "Next" buttons, validation of the data before it's sent and -- more important -- interactivity specific to the application.

You don't want to miss the opportunity to use that third solution because it's the fastest way to get the information you need. Okay, so don't disable the applets, but what *do* you do? As a user you can install the latest version of your favorite browser or you can double the RAM memory and buy a faster microprocessor. As a developer you can write friendly applets that can be suspended and resumed anytime. Read this article to find out how.

Thread Persistence

In my previous article ("Persistent Threads for Friendly Applications," ***JDJ*** Vol. 3, Issue 10) I defined thread persistence and showed why and how it can be used in stand-alone, computer-intensive applications. These applications become friendlier because the user can interrupt them and resume the calculation after an undefined period of time. The state of the threads is saved on disk using serialization API. Thread persistence can be implemented easily for a single independent thread, but

might become complex if you have to deal with many concurrent threads.

There's a big difference between applets and stand-alone applications, however. What does thread persistence for the applets that run in a browser mean? There are two answers...

1. It conserves the state of the threads, between stop and restart, after the applets are downloaded and initialized.
2. It's an essential element if you want to download applets already initialized and customized.

The Java Tutorial (Ref. 1) gives you a number of tips to make the applets more friendly. One is to let users interrupt an applet that executes an animation or something that might be disturbing (e.g., playing sounds). There are a few ways to do that. An old solution is to call the suspend() and resume() methods of the thread that performs the animation. But these methods were deprecated in Java 1.2 because they are deadlock prone. A better alternative is to call the stop() and start() methods of the applet (NOT of the thread). Assuming that you want to interrupt the animation thread in the applet's stop() method, the question is how to assure the persistence of the state of the thread between stop() and start(). The users should be able to resume the animation at the point of interruption.

I have analyzed four ways to make the applets more friendly. The first three don't interrupt the thread, but suspend or commute it in a wait state. The fourth solution interrupts the thread after the stop() method of the applet is called. If the start() method of the applet is called again, the thread is restarted. Conserving the state of the thread in the member variables of the applet guarantees continuity of the animation. The fourth approach thus implements some sort of persistence for the thread that performs the animation, between stop() and (re)start(), after the applet has been downloaded and initialized.

There are two advantages. First, the interruption of the thread frees resources. Note that once the animation is stopped, the users are unlikely to want to resume it. The second advantage consists of the ways to use a second kind of persistence that allows the deployer to customize and then serialize the applet without writing a line of code. A section later in this article ("The Ultimate Use of Thread Persistence") offers details.

BaseFriendlyApplet

The BaseFriendlyApplet class (Listing 1) contains the common code of the next four applets that extend it. I've tried to simulate as simply as possible an applet that performs animation. As the BaseFriendlyApplet class doesn't override the start() and stop() methods inherited from java.applet.Applet, I declared it abstract so that it can't be used in Web pages.

The init() method builds the user interface, whose components are a label and two buttons: "Suspend" and "Resume." The label is the equivalent of a canvas or lightweight component of an animation applet, i.e., it shows the current "frame." (I won't talk about double buffering and sounds; these facilities usually need synchronization, making the thread model more complex.)

The nextFrame() method simulates "the computing of a frame" (it increments the subcounter variable by BIG_NUMBER times). It increments the counter variable and then it calls the setText() method of the label to show "the next frame of the animation." A true animation applet would call its repaint() method instead of setText(). This would generate a PaintEvent, and AWT would call the applet's paint() method. (If your Java interpreter doesn't have a fast JIT compiler, you should delete a "0" from the value of BIG_NUMBER.)

The actionPerformed() method is executed within the AWT drawing and event-handling thread. It's called each time the user clicks one of the applet's buttons. If the user presses "Suspend," actionPerformed() calls the stop() method of the applet. If the user presses "Resume," actionPerformed() invokes the start() method.

Usually, the main methods of an applet -- init(), start(), stop() and destroy() – don't have to be synchronized. Typically, the browser creates a single thread from which it calls these methods in the right order. The subclasses of BaseFriendlyApplet, which override some of these methods, will have to take into account that the start() and stop() methods may be called from two different threads: the one created by the browser for the applet, and the other created for the AWT thread. In addition, the destroy() method may be called, theoretically, while start() or stop() is running within the AWT thread. These methods have to be synchronized so they aren't executed simultaneously within two different threads.

To make testing easier, the "Suspend" and "Resume" buttons remain enabled. A user can click the "Suspend" button twice and the stop() method will be completed twice without an intermediary call of start(). The four subclasses of BaseFriendlyApplet (see below) will have to deal with that.

Using suspend() and resume()

What do you think of a friendly applet that's deadlock prone? This is the case of BadFriendlyApplet (Listing 2), which calls the suspend() method of the java.lang.Thread class.

The applet's start() method creates a thread when it's called for the first time (let's name it the *animation thread*). The next calls resume this thread in case it was suspended by the applet's stop() method. The threadSus-

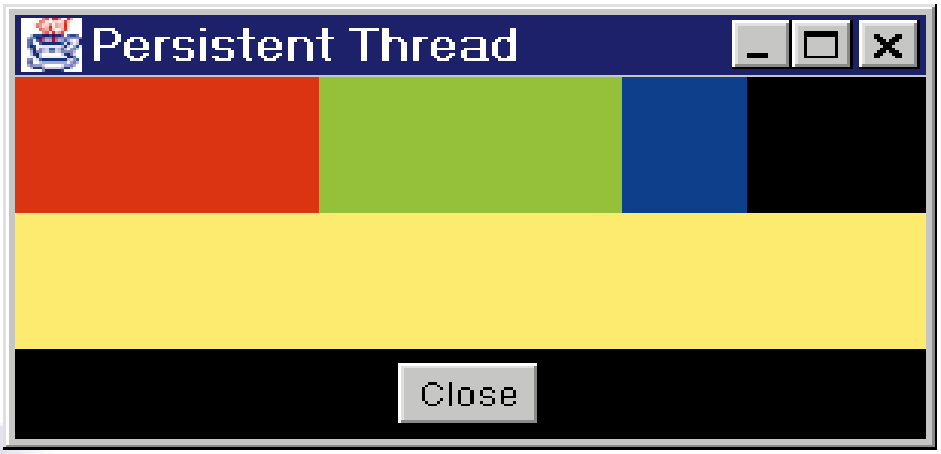


Figure 1: The "Serialize Applet into File" dialog box

pended flag is used to alternate the suspend() and resume() calls. The destroy() method stops the thread. The animation loop is executed within the run() method.

The animation thread may be suspended during the execution of nextFrame() after the user clicks the "Suspend" button. For a real-world applet this means that the animation can be suspended during the computation of the next frame. If the browser asked the applet to repaint itself, that would be a problem even if the applet uses double buffering.

BadFriendlyApplet seems to be stable, but minor changes can lead to disaster. When I added a Thread.sleep(10) call right after thread.suspend(), AppletViewer "performed an illegal operation" (this is an error message from Windows) right before closing. This didn't happen with other browsers. When I made BIG_NUMBER == 0 and clicked the "Suspend" and "Resume" buttons as fast and as often as I could, the applet froze in Microsoft Internet Explorer. Netscape Navigator isn't perfect either. When I modified the applet to allow consecutive suspend() or resume() calls (with BIG_NUMBER == 0), I had to press "Resume" *x* times after *x* clicks on "Suspend" to resume the thread. The methods suspend() and resume() were called from the same thread (the AWT thread), so theoretically the problems shouldn't have appeared.

It isn't worth wasting your time trying to freeze the browsers. You just have to avoid the use of suspend() and resume(). The stop() method of the Thread class mustn't be used either, because it's unsafe. Java 1.2 has deprecated all three of these methods. For details see "Why JavaSoft Is Deprecating Thread.stop, Thread.suspend and Thread.resume" (Ref. 2).

Using wait() and notifyAll()

Is there a solution to writing applets that are both friendly and safe? Of course. In fact, there are many. One of them is based on the use of the wait() and notifyAll() methods of the java.lang.Object class. This section and the next discusses locks and wait sets. *The Java Language Specification* (Ref. 3) dedicates an entire chapter to these subjects ("17 Threads and Locks").

The threadSuspended flag has a new role in the case of GoodFriendlyApplet (Listing 3). When the user clicks the "Suspend" button, this flag will be set to true in the applet's stop() method, which is called from actionPerformed() (inherited from BaseFriendlyApplet). The stop() method will run in this case within the AWT thread. After threadSuspended is set to true, the applet's run() method, which is executed within a different thread (the animation thread), will perform a loop that calls wait() as long as threadSuspended is true.

The wait() method is invoked from a synchronized block. This means that the animation thread has already acquired (or locked) the lock of the GoodFriendlyApplet instance. (Every object has an associated lock that is used for synchronization purposes and that must be locked before the calls of wait(), notify() and notifyAll().)

The wait() call adds the current thread (i.e. the animation thread) to the wait set of the applet, disables the current thread for thread scheduling purposes and releases (or unlocks) the lock. Then it waits for a notification from another thread (i.e., the AWT thread or the browser's thread). (Every object has an associated wait set that represents the list of the threads that have called the object's wait() method and that have not yet been notified.)

When the user presses the "Resume" button, actionPerformed() will call the applet's start() method. This method, which is declared synchronized, will set threadSuspended to false and call notifyAll(). The notifyAll() method will remove the animation thread from the wait set and reenable it for thread scheduling.

After notification, the wait() method can't return the control immediately. Before return, it must reacquire the applet's lock and it can't do that before the synchronized start() method is completed.

An InterruptedException is thrown by any of the Thread.sleep() or wait() calls after the applet's destroy() method invokes the interrupt() method of the animation thread. The exception is caught and the break instruction will transfer the control outside the animation loop. The animation thread will die.

The animation is suspended when the wait() method is called, and resumed after wait() returns the control. The animation can't be interrupted during the execution of nextFrame() unless the browser is closed while nextFrame() computes the next frame, but this isn't an issue.

The pattern of GoodFriendlyApplet may work fine for a simple applet. Some problems might appear in the case of a complex applet from the real world. The next section identifies these problems and suggests a solution.

Using wait() and notify()

What would happen if you used notify() instead of notifyAll()? The user won't see any differences in the case of GoodFriendlyApplet. But if another applet has two or more threads that call the applet's wait() method, the behavior of the applet may depend on the JVM implementation. This is because notify() chooses a single thread from wait set to be awakened, and, as *The Java Language Specification* (Ref. 4) states, "The choice is arbitrary and at the discretion of the implementation." The chosen thread may not be the wanted one. Unlike notify(), the notifyAll() method wakes all the threads from the wait set. Those threads that shouldn't be awakened must call wait() again. This is why GoodFriendlyApplet invokes wait() within a loop. The threadSuspended flag indicates whether the notification has come from where it was expected.

If I declare run() synchronized, the AWT thread will freeze when the user presses "Suspend" or "Resume." The explanation for that is simple. If run() is synchronized, it will keep the applet's lock during its entire execution. When actionPerformed() calls start() or stop(), the AWT thread tries to acquire the applet's lock, which is kept by the animation thread. The user's clicks on the applet's buttons won't have any more effect, and blocking the AWT thread might easily freeze the browser. The worst thing that can happen is the system will crash. Nobody will run the applet a second time. You have to be careful at synchronization not to transform a friendly applet into a hostile one.

The above problems are the consequences of programming errors. The browser can do nothing against deadlocks. Can you avoid mistakes? Yes. You can use different

locks to correctly pair the wait() and notify() calls. BetterFriendlyApplet (Listing 4) shows how. Instead of its own lock, the new applet uses the lock of an object referred by the LOCK member variable. The code for the start(), stop() and destroy() methods is enclosed in synchronized blocks. The applet uses LOCK.wait() and LOCK.notify() instead of its own wait() and notifyAll() methods (inherited from java.lang.Object). It is then no longer necessary to include wait() within a loop. However, the threadSuspended flag is checked twice in the run() method. The first check avoids the acquisition and release of the lock when these operations aren't necessary (the wait() method doesn't have to be called). The second check of the flag is essential because its value may change right after the first verification, before entering the synchronized block.

Using the LOCK variable minimizes the influence of the code that suspends and resumes the animation on the code that implements the logic of the applet (i.e., the animation). A complex applet may now use the synchronized keyword without coming into conflict with the general mechanism that makes the applet friendlier. For example, the run() method may now become synchronized without blocking the AWT thread. A well-designed applet should call wait() after repaint() (within the animation thread), and await a notification from paint(), which runs within the AWT thread). This way, no frame will be lost and no half-computed frame will be shown.

In both cases of GoodFriendlyApplet and BetterFriendlyApplet, the notifyAll() and notify() methods may be invoked before the wait() from run() call if the user clicks "Resume" immediately after "Suspend." Nothing bad happens here because the wait set doesn't contain any threads, and the wait() method isn't called again (threadSuspended is set to false right before notification). In addition, BetterFriendlyApplet could have used LOCK.notifyAll() instead of LOCK.notify(), and the result would have been the same. Finally, the private keyword that precedes the declaration of LOCK doesn't prevent the incorrect use of LOCK's wait(), notify() and notifyAll() methods in regard to purposes other than the implementation of the mechanism that suspends and resumes the animation.

All three of the above solutions (Bad, Good and Better) keep the animation thread alive between init() and destroy(). The next applet shows how to kill this thread between stop() and (re)start(), and demonstrates the advantages.

Persistence between stop() and (re)start()

BestFriendlyApplet (Listing 5) is the applet that implements the persistence of the animation thread. When the user clicks the "Suspend" button, the applet's stop() method calls the interrupt() method of the animation thread. The Thread.sleep() call from run() will throw an InterruptedException. The break instruction will transfer the control outside the animation loop, and the run() method will complete its execution. This means that the animation thread will die, but its state will be kept in the counter and subcounter member variables, inherited from BaseFriendlyApplet. When the user clicks the "Resume" button, the applet's start() method will re-create and restart the thread, and the animation will be resumed from the point at which it was suspended.

The user can also press the "Resume" button right after "Suspend," before the death of the thread. If the animation thread is still alive, the start() method must not create another thread. The correct solution is to cancel the interruption of the thread with the help of the intrCanceled flag. The InterruptedException will still be thrown, but the animation loop isn't broken anymore. The cancellation is canceled when the user clicks "Suspend," "Resume," "Suspend"...

It's unlikely that users will resume animation after they stop it, but they can, if they want.

The animation thread controls the moment when it is killed, so that the nextFrame() method completes its execution before the animation is suspended.

Before discussing why this solution is the best, I'll explain why the initialization of the LOCK member variable had to be changed. If I had declared and initialized LOCK as I've done in BetterFriendlyApplet, the applet's serialization would have failed because java.lang.Object isn't serializable. If I had declared LOCK transient, the object referred by this variable would have been ignored at serialization, and it wouldn't have been re-created at deserialization (instance initializers aren't executed at deserialization). Hence, the start() method would have thrown a NullPointerException. If I had made LOCK class variable (by declaring it static), I would have been mistaken because this variable would have been shared among all of the BestFriendlyApplet instances that would have run at a given moment within the same browser. Probably nothing bad would have happened in the case of this applet, but annoying effects would have appeared if LOCK.wait() and LOCK.notify() had been called. (The thread of an applet could have notified the thread of another applet.) The right solution is to assign a reference of a serializable object to the LOCK variable. The simplest way to do this is with the use of anonymous classes.

```
private final Object LOCK
    = new java.io.Serializable() {};
```

Note that the locks and the wait sets of the objects referred by the member variables aren't serialized because the fields of java.lang.Object don't refer them. The locks and the wait sets are internal data structures managed by the Java Virtual Machine. The programmer can't access them, but you must be aware of their existence to understand thread synchronization and the behavior of the wait(), notify() and notifyAll() methods.

The pattern of BestFriendlyApplet offers two important advantages. First, the thread created in the start() method is interrupted after the call of the stop() method. Hence, the resources allocated to the animation thread are released. Again, it is unlikely that the users will resume the animation after they stop it. (But they can resume it, if they want.) The second advantage is that the applet is serializable after the stop() method is called. The other applets (Bad, Good and Better) aren't serializable because the java.lang.Thread class isn't. The following section gives more details about the second advantage.

The Ultimate Use of Thread Persistence

Java 1.1 has extended the <APPLET> tag, so that instead of the CODE attribute, you can use the OBJECT attribute, whose value must be the name of a file that contains a serialized representation of an applet. After it downloads this file, the browser will deserialize the applet and call its start() method. (The init() method isn't invoked.) This allows the deployer to offer many customized versions of the same applet without writing a line of code. Such a deployment might be useful for the complex applets, whose customizations need something more than the parameters of the <APPLET> tag. You have to be aware that Netscape Navigator and Microsoft Internet Explorer don't yet recognize the OBJECT attribute. Hence, to use this feature of Java 1.1, you will have to use Sun HotJava or AppletViewer. (You may use any Java 1.1-compatible browser, including Explorer and

Navigator, to run the applets presented in this article as long as you only use the attributes of the <APPLET> tag from Java 1.0.)

How does the OBJECT attribute work? An example is the best answer for this question.

1. First, you must create an .HTML file (e.g., CodeAttrib.html) that uses the <APPLET> tag with the CODE attribute.

```
<applet code = "BestFriendlyApplet.class"
width=200 height=200> </applet>
```

2. Next, run "AppletViewer (e.g., CodeAttrib.html.)
3. Let the applet run a while, then select the "Stop" item from the Applet menu of AppletViewer. AppletViewer will then call the applet's stop() method. You'll have to wait until the animation thread is interrupted.
4. Select the "Save..." item from the Applet menu. AppletViewer will show a dialog box, whose title is "Serialize Applet Into File" (see Figure 1).
5. Type, for example, "aBestFriendlyApplet.ser" (without quotes) within the "File name" text field, and click the "Save" button. The dialog box will be closed, and AppletViewer will serialize the applet in aBestFriendlyApplet.ser file.
6. Close AppletViewer.
7. Create a second .HTML file (e.g., ObjectAttrib.html) that uses the <APPLET> tag with the OBJECT attribute.

```
<applet object = "aBestFriendlyApplet.ser"
width=200 height=200> </applet>
```

8. Finally, run "AppletViewer ObjectAttrib.html" or download ObjectAttrib.html in the HotJava browser. AppletViewer/HotJava will deserialize the applet and call its start() method. (The init() method isn't invoked.) The animation will be resumed.

One small problem is that if you select the "Save..." item and click the "Save" button (steps 4 and 5) immediately after "Stop" (step 3), the animation thread may still be alive. If so, its associated Thread object is referred by the thread member variable of the applet. AppletViewer will print an error message to the console because the java.lang.Thread class doesn't implement java.io.Serializable.

```
in appletSave:
java.io.NotSerializableException:
java.lang.Thread
```

Wait a little and then try again to serialize the applet, using the "Save..." item. (Repeat steps 4 and 5.) This inconvenience is minor because the deployers serialize the applet before they insert it into public Web pages. It's incorrect to declare the thread variable transient to avoid the error message because this would allow the serialization of the applet during the computing of the next frame. Remember, the applet encapsulates the state of the animation thread. You have to let this thread arrange its own death and store null in the thread member variable.

This is an interesting trick. The applet can't be serialized before the execution of "thread = null;" because the Thread class isn't serializable. Nevertheless, the applet can be serialized after the thread variable is set to null (in the run() method, before break) because the Serialization API doesn't look at the type of the member variables, but at their values.

The applet's stop() method can't wait for the death of the animation thread. If it did this, the user wouldn't be able to cancel the suspension of the animation because the AWT thread and thus the "Resume" button would be blocked during the wait period of stop().

> *"You are the one who decides what's best for your applets."*

Applet Persistence in the Real World

Imagine the following scenario. You are developing an applet that implements a neural network, and you want to show visitors to your site how the results have improved during training. From time to time you stop the applet and serialize it. Then you restart the applet and training continues. After a while, the network reaches its optimum, and then will begin to forget what it has learned. What you have to do is insert the serialized variants of the applet that were saved before and right after the optimum was reached into a Web page. This technique can be used for any kind of applet that shows the evolution in time of a phenomenon, and you won't have to write I/O code or develop multiple versions of the applet.

The above example is a specialized one. The pattern of BestFriendlyApplet is very general and has a double role: to make the applets both friendly and serializable. The latter might not interest you right now, however, because Navigator and Internet Explorer don't yet recognize the OBJECT attribute of the <APPLET> tag. There is one more disadvantage: the .ser files increase the download time. You can use the transient keyword to control what is serialized, but the objects referred by the member variables that your applet inherits from java.applet.Applet will be serialized. Among these objects are the AWT components of the applet.

In addition to simplifying customization, there is one more advantage: the init() method isn't called anymore, so a fast initialization comes after the slow download.

You are the one who decides what's best for your applets. Even if you don't use serialization, you can still make the applets more friendly without performance costs and without limiting the number of target browsers.

Summary

This article has answered many questions: What could applet persistence mean for the real world? How can we implement persistence for threads and applets? How do we write thread-safe friendly applets? How can we pair the wait() and notify() calls? What's the difference between notify() and notifyAll()? Why not use suspend() and resume()? But this article is not just a list of questions and answers. Starting with a nonserializable deadlock-prone applet, I've identified the problems, found the solutions and designed a pattern for serializable friendly applets.

In my next article I'll discuss the persistence of the Swing components.

References

1. *Mary Campione and Kathy Walrath*, The Java Tutorial, *Addison Wesley.* ***java.sun.com/docs/books/tutorial/***
2. *Sun Microsystems, "Why JavaSoft Is Deprecating Thread.stop, Thread.suspend and Thread.resume."* ***java.sun.com/products/jdk/1.2/docs/guide/misc/threadPrimitiveDeprecation.html***
3. *James Gosling, Bill Joy and Guy Steele*, The Java Language Specification, *Addison Wesley.* ***java.sun.com/docs/books/jls/***

About the Author

Andrei Cioroianu, an independent Java developer, has a BS in mathematics/computer science and an MS in artificial intelligence. His focus is on 3D graphics (Java 3D), software components (JavaBeans) and user interface (AWT, JFC). You can reach Andrei at andcio@hotmail.com.

JColor

A widget that satisfies advanced and novice software needs

by **Claude Duguay**

This month we have a colorful widget for you. While the JFC provides a pretty nice color picker, it doesn't seem to go the extra mile that users of imaging software have come to expect. Once exposed to that software, some users have become pretty sophisticated and you have to use a well-designed color selection control – should you need one in your application – to make a good impression on them.

This column explores a widget called *JColor* that provides six views on the color spectrum in a single, easy-to-use interface. Figure 1 shows the JColor control in action, with the Blue view activated. The six views correspond directly with the red, green and blue of the RGB model and the hue, saturation and brightness of the HSB model.

The RGB and HSB color models are based on three dimensions, something often difficult to visualize on a flat display. Our approach uses the two dimensions of a rectangle and an additional dimension in a vertical gradient.

The view selected by the radio controls of our interface determine which dimension is displayed in the vertical gradient. The remaining two dimensions are represented by the two axes of the rectangle. You can pick colors graphically in any model, or type the RGB or HSB values directly in the fields.

Image Producers

Both the ColorSliderImage and ColorMatrixImage classes are implementations of the ImageProducer interface. An image producer implements a minimum interface that allows it to dynamically produce arbitrary images that can be read by an ImageConsumer and displayed by an ImageObserver.

The JFC offers a simple abstraction to the ImageProducer class called *SyntheticImage*, which provides a constructor that needs to know the width and height of the image and a computeRow method to actually produce the pixels one row at a time. This is very convenient, since little coding is actually required to produce an image.

We implement two image producers. Both of them calculate a color gradient vertically and one of them calculates an additional horizontal gradient. In both cases, we use a set of constants that tell us which of the six view modes we're in. The constants, presented in Listing 1, are RED, GREEN, BLUE, HUE, SATURATION and BRIGHTNESS. The images are produced dynamically through the interface.

Listing 2 shows the source code for ColorSliderImage, which produces a vertical gradient based on the current style. For the RED, GREEN and BLUE styles we create a gradient between zero and the specified color, representing a value between zero and 255. The HUE, SATURATION and BRIGHTNESS values are determined by using the static HSB to RGB method in the Java Color class.

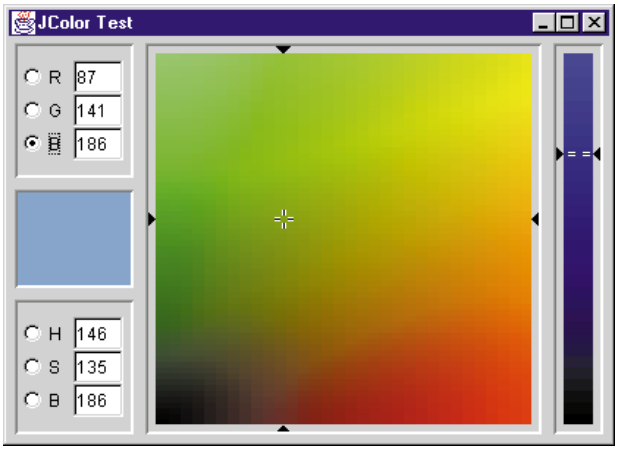


Figure 1: JColor in the Blue view

Listing 3 shows the code for ColorMatrixImage, which produces a vertical and horizontal gradient covering the full two-dimensional range specified by the current style. Notice that the ColorSliderImage actually represents the selected style, so ColorMatrixImage produces the remaining two dimensions. If, for example, the RED style is selected, the ColorMatrixImage will actually represent the green and blue color components.

Color Selectors

Two widgets need to be implemented to support interactive color selection. The JColorSlider and JColorMatrix controls use the image producers we developed to display the range of values available for each selection. Two visual cues are employed to show the user the current selection – arrows on the outside of the color area and a crosshair surrounding the current position.

The arrows are drawn as triangular polygons on the border of the control. The JColorSlider control provides arrows on the left and right of the current position. The JColorMatrix control draws left and right arrows, along with top and bottom arrows. To make all this easy to follow, we provide separate methods for each of the arrow drawing routines.

The crosshair is designed to handle the unpredictable underlying color spectrum. If the underlying color is dark, a black crosshair would be ineffective. If we make the crosshair white, the same problem occurs when the underlying color is too bright. The solution is to use a combination of black and white with a black central crosshair and a white edge on each side of the black lines.

Listing 4 shows the AbstractColorSelector class from which both JColorSlider and JColorMatrix inherit. It encapsulates basic code for handling common member variables along with action listener registration and event handling. Empty MouseListener and KeyListener methods are also provided; since we're primarily interested in the mousePressed and keyPressed events, we can ignore the others.

Listing 5 shows the code for JColorSlider. We set the style and register to receive mouse, key and focus events. Besides setting the style member variable, the setStyle method sets the image to null before repainting, forcing a new image to be generated by the ColorSliderImage producer. The paintComponent method draws the image, arrows and crosshair,

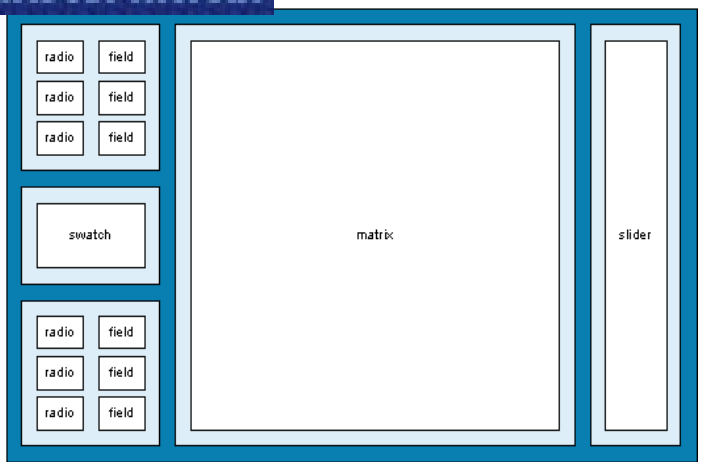


Figure 2: JColor nested panel layout

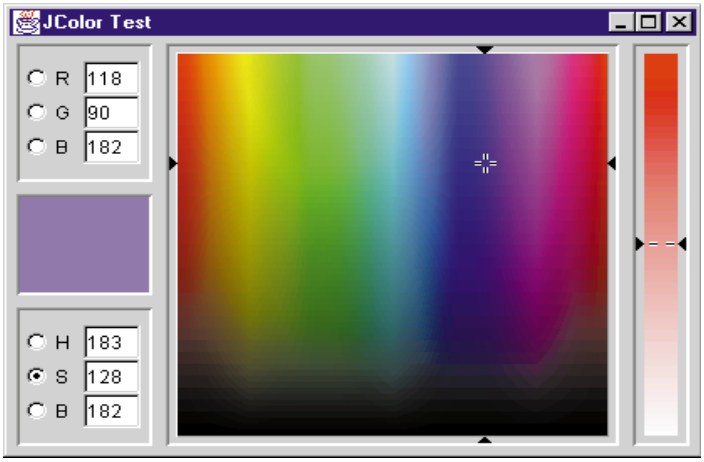


Figure 3: JColor in the saturation view

and the focus rectangle when appropriate.

The setYValue and getYValue methods handle setting and getting the vertical position. Internally, we calculate the actual pixel location. From the outside these values are expected to be between zero and 255. We also set the preferred size so that window packing will create ideal dimensions for the control, with a pixel for each discrete position. The object can be scaled as needed, however, so this is considered a guideline.

Listing 6 shows the code for JColorMatrix, which extends the JColorSlider control. All the vertical handling is identical, so we extend the JColorSlider behavior to handle horizontal positions. We add the setXValue and getXValue methods along with top and bottom arrows, and modify the code for paintMethod and drawCrosshair to accommodate the additional dimension. The getPreferredSize method returns symmetrical, ideal dimensions for a square area with a single pixel for each discrete unit. *Note*: If you make it smaller, you'll lose some resolution, and making it larger repeats pixels, so the preferred size is highly recommended.

The JColor Control

The main JColor control is implemented as an extension to JPanel for flexibility. Using this strategy allows us to place it in any component or window. This is the most complex class in this collection, primarily because the JColor panel handles all the button, field and selector events, and coordinates the six views provided to display more than 16 million (24 bit) colors in the spectrum.

Figure 2 shows how the internal panels and components are arranged. The JColor constructor creates each of the instances and stores a reference to the buttons, fields and color selectors in member variables.

Let's quickly review behavior in the JColor widget. If the user selects one of the radio buttons, the matrix and slider view images are updated to reflect the color selection model. When the user types in a value directly in one of the fields, we clip values within the zero to 255 range. If the field being edited is in the RGB range, we automatically switch to one of those views if one isn't already active. To be consistent, we pick the view associated with the current field. The same is true of entering values in the HSB fields if one of the views isn't active already. A value changed in the matrix or slider component is immediately reflected in the field values and the swatch panel, both of which keep in mind which model is active at the time.

> *"The JColor widget provides developers with a color selection control that's both comprehensive and flexible"*

Most of the work is done when action events are triggered. Listing 7 shows the code for the actionPerformed method, with most of the field handlers omitted. The red field handler is sufficiently indicative of how the others work to get the idea.

The radio buttons are handled first. In each case the style value changes and a condition that checks for RadioButton events follows immediately to switch the slider and matrix styles dynamically. No matter what the source of the event, we call setColorText to update the field values and repaint the swatch to reflect the current color.

Listing 7 shows the condition that handles the red field events. We first check to see if one of the RGB radio selections is active. If none of them are, we switch views by setting the style value and updating the matrix and slider styles. In either case, we check to see which context is active and, then set the appropriate *x* or *y* value in the slider or matrix controls. The same mechanism is applied to each of the fields in a set of subsequent conditions.

Listing 8 shows code for JColorTest that demonstrates usage. We create a JFrame and watch for the windowClosing event. The code puts a new JColor panel into the center and sets an initial color with the JColor.setColor method. We fetch the color with getColor on exit. Figure 3 shows the JColorTest running. Notice that the JColorMatrix control has the focus and appears slightly lifted from the page.

Summary

The JColor widget provides developers with a color selection control that's both comprehensive and flexible. Designed to handle a wide variety of needs, it presents itself as an alternative to the JColorPicker control provided with the JFC. More important, it provides a sophisticated design that satisfies advanced imaging software needs as easily as the needs of a novice user.

CODE LISTING

The complete code listing for this article can be located at **www.JavaDevelopersJournal.com**

About the Author

Claude Duguay has been programming since 1980. In 1988 he founded LogiCraft Corporation, and currently leads the development team at Atrieva Corp. You can contact him with questions and comments at claude@atrieva.com.

claude@atrieva.com

Let Age Approve of Youth

Help Wanted: Java Developers Only

by **Alan Williamson**

Another month has rolled by and here we are, entering a brand new year. Whether it's just me getting older or the earth speeding up, time seems to be flying past at a tremendous pace. This column, for example – would you believe? – is now celebrating its six-month anniversary. That sure came around quickly. Before you know it, people like us will be known as the veterans of the Java industry, considered the early adopters and forging on ahead with the new technology. Exciting when you think about it.

The column has taken us many places; we've explored many different issues, thrashed out some interesting ideas and even had some fun along the way. I hope you've enjoyed reading the beast as much as I've enjoyed composing it. Let's hope the next six months are as kind to us. Let me thank all of you who've taken the time to e-mail me. You know who you are! It's much appreciated and I enjoy reading and answering your e-mails, so please, keep them coming.

And now, on with this month's look at the Java universe.

This month I'm going to go down the route of employment, because here at N-ARY, we're going through the painful process of recruiting. As usual, I'm going to analogize my findings with a human personality trait – this month I'm going to go for loyalty. But I'll come back to that in a moment.

We're expanding, and that brings more work with it. We need more bodies. Not a huge problem, one would have thought, to go and hire a couple of Java developers. Boy, were we mistaken! We never realized the minefield we'd be entering. Since we're based in the UK, we began our search on home soil.

First Things First

The first thing we prepared was a job specification. A good place to start. It was quickly established that there was a junior role that would in time develop into a more senior position. So we were looking for someone just out of university, or not long in industry. Due to the budget we had allocated for this new person and the nature of the work, a graduate fitted the bill perfectly. What we did insist on was someone with Java experience. Whether it was coding at university or during spare time, we needed someone who could code from day one, as opposed to our training them. Considering the claims from Sun concerning the number of Java developers in existence, we didn't think we were asking too much.

We began our search by first going to the place where graduates are supposed to hang out: universities. We e-mailed all the major universities in our country and got nowhere fast. Why? Wrong time of the year. Our search began around August of last year, but I was sure some graduates must still be lurking around after the exams. If they were, none presented themselves.

Around this time, I began to read many articles about the skills shortage the IT industry was experiencing. In fact, there was even a move in this country to train prisoners to deal with the Year 2000 problem. I'm not quite sure what happened to that scheme, but all seems to have gone quiet on that front. Somebody in our government must have thought it a good idea at the time. Bless.

While this shortage was being reported, another irony was unfolding. Everywhere you looked, another major corporation was laying off staff. Not just one or two people, but thousands. A quick trawl through c-net.com showed the full horror of the situation. According to the news reports found at C-NET, we have the likes of Nortel laying off 3,500 employees; Netscape, 300; SGI, 1,000 – and even our Japanese friends, with Hitachi laying off 650. These are just the big household names; I'm not even listing all the smaller companies that are getting rid of 20 to 50 people. It looks somewhat bleak.

But I have to ask: When so many are joining the job market, how come we're still experiencing a shortage? There are a number of possible reasons. First of all, the job cuts may be of nonskilled workers. This is possible, but I know some people that have left the ranks of Nortel, for example, and they are far from nonskilled. So let's assume it's not all administrative staff that's been removed. Besides, administrative people traditionally don't cost that much when compared to a highly trained developer. If the job cuts are made to save money, then removing a team of developers as opposed to a number of secretaries will save more. When the accountants need to make cuts, they look at the higher end salaries and begin with them, then generally work up. Stands to reason – getting rid of one person as opposed to the equivalent of three will keep morale higher and not look as bad to the press.

The assumption is that there are people now looking for jobs. Of course, if there is a skills shortage, but companies are making major job cuts, it begs the question of which companies are looking to hire. But let's not deal with that one just yet.

Okay now. On one hand we have the claim that says we're suffering a skills shortage; on the other we're making significant job cuts. Maybe the two are related. Maybe the reason there are so many job cuts is because the skills the company is looking for aren't actually in-house. This would make sense to some degree. But again, a question about retraining the said personnel raises its ugly head. Surely that's got to be cheaper than going through the whole firing and hiring loop.

Reality Check

But there's another possible reason that's a bit more controversial. What if there are enough bodies and they claim to have the skills, but when these people are hired companies find they've been duped? They discover the level of expertise isn't quite what they expected. Looking at the Java universe we can see this is very evident. A number of people claim to know Java, but when you look closely at their CVs you discover, for example, an HTML developer with no formal programming skills. Call me cynical, but a programmer that does not him/her make.

In our quest for Java developers we've seen many of these CVs. Most are worthless. We need a developer, a software engineer. We don't need another HTML body. We need someone who knows algorithms, someone who knows one end of a class from another. Sadly, the self-taught brigades aren't up to scratch.

But why is there a skills shortage? Why are so many people not trained for the jobs

Object Matter

www.objectmatter.com

the industry is looking to fill? Is it because companies have taken on too much work? Have they oversold themselves? Who knows? A general slowdown of development wouldn't go amiss, and regular readers of this column know that I'm all for a general slowdown of Java to allow the rest of the world to catch up.

Where Is Everybody?

Taking this into account, our search for souls was getting nowhere fast. We then looked at other available resources – the recruitment agencies. This turned out to be fun. At this point I'd like to curse the person or persons who felt it was a good idea to name the HTML scripting language "JavaScript." Do they realize the amount of confusion and heartache this has caused the industry?

We sent our job specification to the agencies and instantly our inbox began to fill with potential candidates. We were excited. At last, potential N-ARY employees were coming in! Our initial excitement was soon to dampen, however, as we read JavaScript over and over again. These aren't Java developers! What's going on here?

We phoned some of the agencies. We said, "Thanks, but no thanks. You haven't sent the right sort of candidates."

"But we did," came the answer. "You mean Java has nothing to do with JavaScript?"

A learning curve is still to be taken by some agencies, it would appear. Which is frightening when you think about it. Companies are trusting such agencies to be their recruitment agents. If anybody should know the difference, they should!

Once the difference was pointed out, the inbox didn't get quite the same amount of attention. And the CVs that did come through were not that great but still felt the need to ask for huge amounts of money, which I found highly amusing.

Don't get me wrong. We have no problem paying for good people. As the old saying goes, "Pay peanuts? Get monkeys." But if we have to pay for the poorly skilled, it staggers the imagination to think what highly skilled people want.

So what seemed to be an innocent enough task – to hire a couple of bodies – was turning out to be as difficult as the quest for the Holy Grail. Exasperated, we decided to look for developers beyond the bounds of our own country. We've used the services of PSI Limited in India. This large Indian development house, run by one Mukesh Patel, did us proud with a number of Java projects so our faith in overseas developers was high.

We quickly updated our job specification to include free accommodation, and sent it off to various universities around the globe. The beauty of the Internet meant this wasn't that big a task. Well, what a difference that made!

Not only did we get CVs in, but they were of a very high standard, complete with examples of work and references. They liked the salary, they liked where they would be working and they were extremely enthusiastic. The upshot? We hired a recent doctorate from Thailand who majored in Java Servlets and JDBC, and we're still choosing another from a large pool of CVs.

I think the whole thing boils down to money. There is always a discussion of how the Asian countries are polluting the industry by driving down salaries. I think this is a good thing, not a bad thing. I personally feel we are sometimes overpaid, and as a by-product some of us get complacent. We stop trying. We know we're in demand, and if we don't get on with management or if we do something wrong, we know we'll be snapped up again, probably with a pay rise.

Our industry suffers from a high staff turnover rate. Sometimes I feel I have the kiss of death with regard to people. In the last year around 70% of all the people I have built up a rapport with have left their companies and moved on. It's funny on the one hand but extremely frustrating on the other. Surely this continual moving about can't be doing the industry as a whole any good. Something has to give.

Back to Loyalty

Back to our trait of the month, loyalty. Are people no longer loyal to their companies? When the going gets tough, it's too easy to move. In our world a company's greatest asset is its employees. We develop virtual products, software, which has to be maintained and further enhanced. Changing the team all the time isn't healthy, for the company or the end client. I don't think it's a case of the company needing to try harder to keep their staff. After all, there's a limit to the salary you can pay one person.

So I welcome this new influx of people that are keen, enthusiastic and above all looking to work for the love of it as opposed to the dollar. That is, of course, until they discover how the world operates and we lose them, and have to start this whole process over again!

About the Author

Alan Williamson is CEO of N-ARY Limited, a UK-based Java software company specializing solely in JDBC and Servlets. He recently completed his second book, which focuses on Java Servlets. Alan can be reached at alan@n-ary.com (www.n-ary.com).

alan@n-ary.com

JProbe Profiler 1.1.1

by KL Group Inc.

A profiling and analysis tool for Java

by **Jim Milbery**

The Java language removes several problem areas for developers, compared to C++ development, by its elimination of memory allocation and pointer management. While this generally makes Java programs more stable than their C++ brethren, it can often lead to the misconception that Java programs don't need to be optimized or profiled. Nothing could be further from the truth, especially when you consider that Java is being used to build enterprise-class applications, which require high performance. KL Group has built a strong reputation for supplying Java components to many of the leading software vendors, and they have extended their Java presence with JProbe Profiler.

Product Installation

KL Group delivers the software directly from their Web site for download, as well as offers the software on CD-ROM. JProbe is available for Windows 95/NT and installs with an InstallShield executable. I was able to get the software installed in a matter of minutes, and the complete installation required only about 14 MB of disk space. JProbe requires the JDK 1.1.5 version to run, but you can profile any application that was written for either the JDK 1.0 or 1.1 series virtual machine. If you have already installed the JDK 1.1.5 virtual machine, you'll still need to install the version that comes with JProbe, since it's a specialized version of the standard 1.1.5 JDK virtual machine. The runtime environment for JProbe allows you to set CLASSPATH definitions for individual programs as needed, so it's not necessary to modify your Java development environment to get started.

Uncovering Performance Bottlenecks

Java as both a platform and a language offers a number of advantages in terms of portability and standardization. Programmers across the globe are migrating to Java from a variety of other languages and platforms. For many of us the move to Java may also be our first real, hard-core experience with object-oriented programming. As a result, the programs you write may have hidden performance bottlenecks that slow down processing and eat memory. KL Group's JProbe is designed to uncover these hidden performance anomalies including such common problems as excessive object creation, method calling and thread creation, and inefficient memory usage. The Profiler collects timing information and memory data as you run your Java programs, and JProbe supports most of the leading Java development environments. I used Oracle's JDeveloper product to create a few sample Java programs for testing. Once you've compiled your code in your favorite development tool, start the Profiler and select the "Run" menu. I was quickly able to search for possible performance problems in my code by using the "Memory Usage Monitor," as shown in Figure 1.

The memory monitor charts memory as it's used by the program and is generally the first place you start when you use the Profiler. KL Group includes a number of sample program runs with JProbe, and I'd advise you to start by using these examples with the Profiler's online help to get an overview of the Profiler's capabilities. You have a choice of running a program through to conclusion as one long run, or you can divide the performance data into specific chunks by using "snapshots." I found the snapshots invaluable as a tool for comparing the first pass of a program with subsequent executions through the same set of code. They were especially helpful tracking down problems with event-driven code. I was able to use the memory monitor to drill down into details of my code quickly and easily. The snapshots can be used to track calling relationships, as shown in Figure 2.

You can see in the diagram that the profiler can show a hierarchical display of method calls, which can help you expose the most expensive methods. If you need to, you can click down to the source code directly from the diagram, and the graphical interface uses colors to highlight the more expensive method calls. If you choose to view the source within the Profiler, you can see how much resource is used by each routine in a panel next to the source window. The Profiler makes it easy to shift between multiple snapshots, and you can save the entire set of snapshots and program definitions for reuse at a later time. I was impressed with how quickly I was able to find problems in my own programs, but I'll take a pass on telling you just how bad my coding actually was before JProbe got ahold of it!

Performance and Usability

KL Group claims to have improved the performance of JProbe Profiler by a factor of 10 with this release. I didn't test any large or long-running programs with the Profiler, but JProbe ran briskly enough on my development platform. I couldn't see any easy way to test programs out in batch, although I was able to save test ses-

JProbe Profiler 1.1.1

KL Group Inc.

260 King Street East

Toronto, Ontario, Canada M5A 1K3

Phone: 800 663-4723 Fax: 416 594-1919

Web: www.klg.com

E-mail: direct@klg.com

Requirements: Microsoft NT 4.0 or Windows 95 on a Pentium-class processor. At least 32 MB RAM. Approximately 40 MB free disk space. Since the necessary JDK environment is included internally with the product, there is no need to install the Java Development Kit.

Price: $499/developer

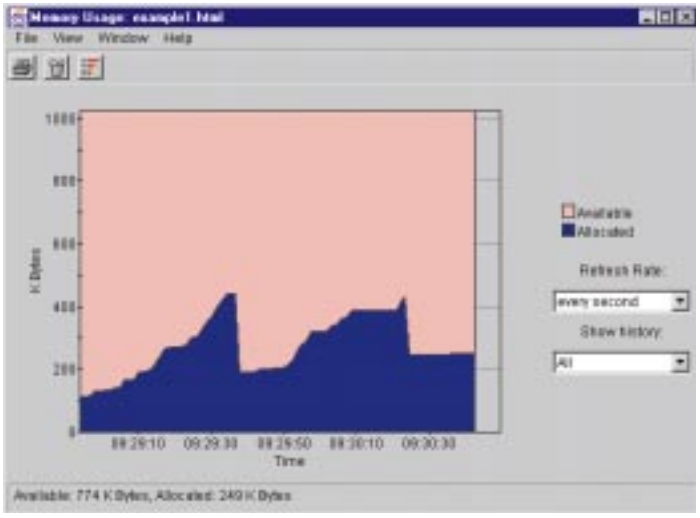


Figure 1: Memory usage monitor

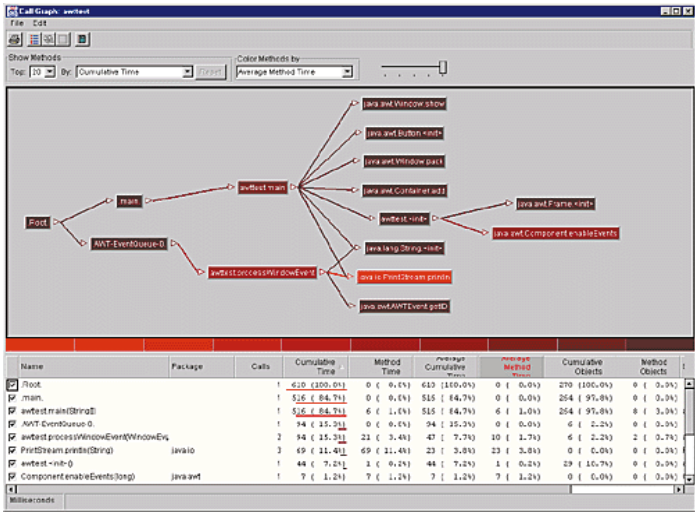


Figure 2: Track calling relationships

sions for later analysis, and you can save programs for reexecution as well. The online documentation with JProbe Profiler is adequate for using the product, but it's a little weak in the area of interpretation. The help files have a tendency to expect that many of the performance numbers will be self-explanatory, and novice programmers may find that it'll take some practice to get a handle on interpreting results. I'd encourage you to make use of the tutorial and sample files before tackling any of your own code.

Final Thoughts

There are several products on the market that purport to provide performance profiling for Java. However, when I quickly searched the various Java newsgroups for user opinions, I found that many programmers had good things to say about JProbe Profiler. One programmer in particular claimed that JProbe had helped him to get a hundred-fold improvement in performance. My informal Web survey seemed to indicate that JProbe is clearly the leader in Java profilers, and from my brief experience with the product, I'd be inclined to agree.

About the Author

Jim Milbery is an independent software consultant based in Easton, Pennsylvania. He has over 15 years of experience in application development and relational databases. Jim can be reached at jmilbery@milbery.com or via his Web site at www.milbery.com.

www.milbery.com

You May Shed a Tier or Two Trying to Choose One

So You Want a Stand-alone Database

by **Tim Callahan**

When you see the words *Java database application*, you probably start thinking about enterprise-level solutions with multitier architectures and distributed deployment. But Java is a great general-purpose, object-oriented language and thus a good choice for developing smaller scale, stand-alone database aplications as well. You can enjoy the benefits of programming in Java regardless of an application's scale or deployment.

I define a stand-alone database application as one that is installed and maintained primarily by the end user. Deployment may be on an isolated computer or small network for shared database access. Examples of stand-alone database applications are numerous in shareware, consumer software and general-purpose business programs.

When it comes to finding a stand-alone database solution for Java, there's good news and bad news. The good news is that solutions are available. The bad news is that you may have difficulty choosing the best one. No single solution may meet all your needs, and many products are just now emerging. Several Java-related issues such as application tiers, JDBC, portability and RMI further complicate the situation. This article will help you choose a stand-alone database solution for Java.

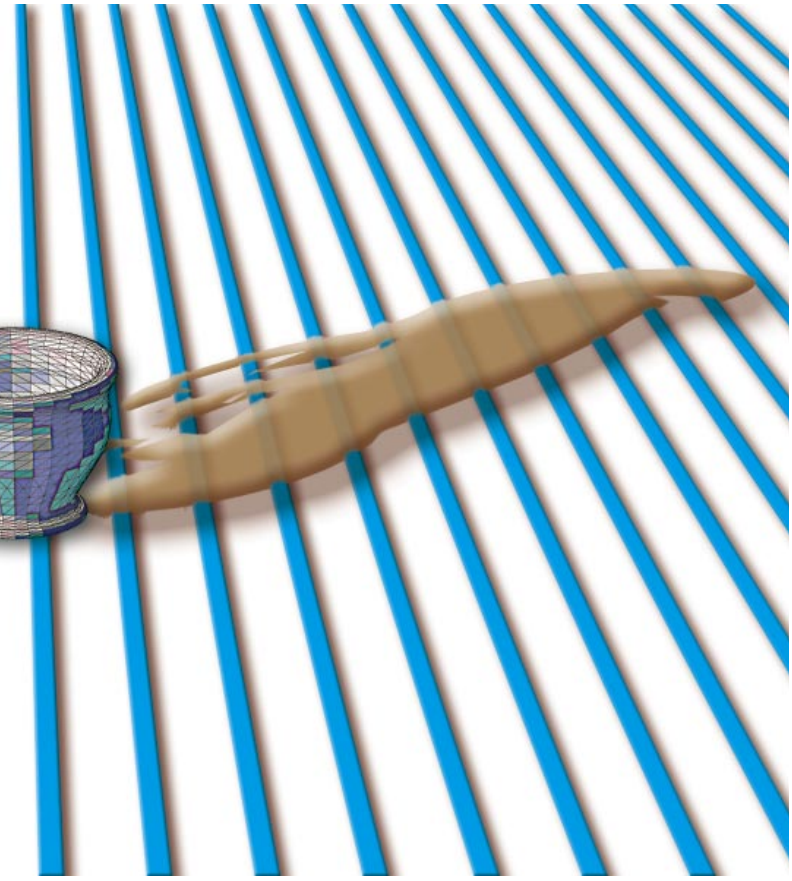
A Java Database Primer

Before we continue, let's review some concepts relevant to the use of databases with Java, including database types, application tiers, JDBC and RMI.

Database Types

One way to classify databases is by their structure and functionality. Common structures include flat file, navigational, relational, object-oriented and object relational. Functionality in a database usually comes from its structure and any associated database management system (DBMS). A database's structure provides functionality such as data types, records and indexing. A DBMS provides significant functionality aimed at improving consistency, reliability and performance.

- Flat file databases store information in an unstructured fashion and usually have little added functionality.
- A navigational database structures data using fields, records and tables. The structure allows easier access and adds functionality such as data types and indexes. Navigating the database and enforcing relational rules is the responsibility of the application.
- A relational DBMS (RDBMS) structures data based on relational rules and usually provides other functionality such as a query language, stored procedures, triggers and performance tuning.
- An object-oriented DBMS (OODBMS) doesn't impose a relational structure on data. Instead, objects are stored as objects and the relationships between objects are preserved. Object-oriented databases can increase development productivity and provide other benefits.
- An object-relational DBMS (ORDBMS) combines the strength of the relational structure of an RDBMS with the object orientation of an OODBMS. It is usually implemented as an RDBMS with OO



extensions such as the ability to use classes as column types, to access object methods and fields in queries, and enhanced SQL.

Application Tiers

An application's deployment architecture is often described by its number of tiers. Work is done on each tier, with certain tasks usually performed on specific tiers. Allocating tasks to tiers often depends on whether they implement application logic, business logic or database logic. Communication between tiers is typically over a network.

Multiple tiers provide a way to advantageously partition and distribute the tasks in a database application to reduce client maintenance costs and improve performance. These improvements come at the cost of a more complex operating environment, however, since each tier may require special software or additional administration. Figure 1 illustrates typical tiered architectures.

- In a single-tier architecture (common in navigational databases) the application handles all processing and directly accesses the database. Application and database logic are tightly coupled and usually no software except the application is required.
- A two-tier architecture splits the processing between the application and a DBMS. The DBMS handles database access and provides functionality such as integrity constraints, triggers, stored procedures and performance tuning. This classic client/server architecture partially decouples the application and database logic.
- A three-tier architecture attempts to completely decouple the application, business and database logic. The application logic is restricted to a presentation layer at the client, objects in the middle tier encapsulate business logic and the third tier provides database access. This architecture is typical in distributed systems and is common with Java.

JDBC

JDBC (Java Database Connectivity) is an abstraction layer defined by JavaSoft that provides a standard SQL-based interface to any data source. You write your application in terms of JDBC classes and methods and a JDBC driver handles the interface to the actual data source. This provides an opportunity to decouple applications from the database implementation since all data access uses the JDBC API. A solution is JDBC-compliant if it implements the JDBC API and can pass JavaSoft compliance tests. An important component of any JDBC solution is the driver that handles the database interface. JavaSoft classifies JDBC drivers into four types (see Figure 2), although not all possible drivers fit neatly into a category.

- **Type 1** – *JDBC to ODBC Bridge:* This driver translates JDBC method calls to ODBC function calls and provides access to any ODBC data source via JDBC. Multiple levels of translation can slow performance, and deployment is complicated because of the ODBC driver required on each client computer. Multiuser access is complicated since many ODBC drivers are not networked
- **Type 2** – *Native API, Partly Java Driver:* This driver translates JDBC into calls to a native database API. Performance is generally better than with Type 1 drivers because of one less translation layer. Deployment problems remain since Type 2 drivers require the database vendor's proprietary library on each client computer. Classic two-tier applications often use this type of driver.
- **Type 3** – *Network Protocol, All Java Driver:* This driver translates JDBC calls into a DBMS-independent network protocol that a middle-tier server translates into a DBMS-specific protocol. This flexible driver is well suited for distributed three-tier architectures. Performance can be slow because the middle tier may itself use a Type 1 or Type 2 driver to access the database. The additional tiers make deployment complex and the network protocol may be proprietary even though it's database independent.
- **Type 4** – *Native Protocol, All Java Driver:* This driver converts JDBC calls directly into the network protocol used by a specific DBMS. Native database calls are made directly over the network so performance is usually good. The database vendor usually supplies this type of driver since native DBMS network protocols are generally proprietary.

Note that not all drivers fit into one of these four types. Consider, for instance, a native API driver that is 100% Java and directly accesses a local database. This type of driver is well suited for access to desktop databases but doesn't fit into one of JavaSoft's defined types. Some vendors do supply unique JDBC drivers that provide advantages in certain scenarios.

RMI

Two Java applications can communicate over a network using a standard technique called *Remote Method Invocation* or *RMI*, which is defined by JavaSoft and is available free. As its name implies, RMI lets a client application invoke the methods of objects running on a remote server.

Shared database access is often implemented with RMI, so an overview is in order. To use it, first create a public interface (called the *remote interface*) to define the remote methods. Then create classes that implement the remote interface, called (not surprisingly) *implementation classes*. The remote objects are instances of these classes running on the server. Next, use JavaSoft's RMIC compiler to

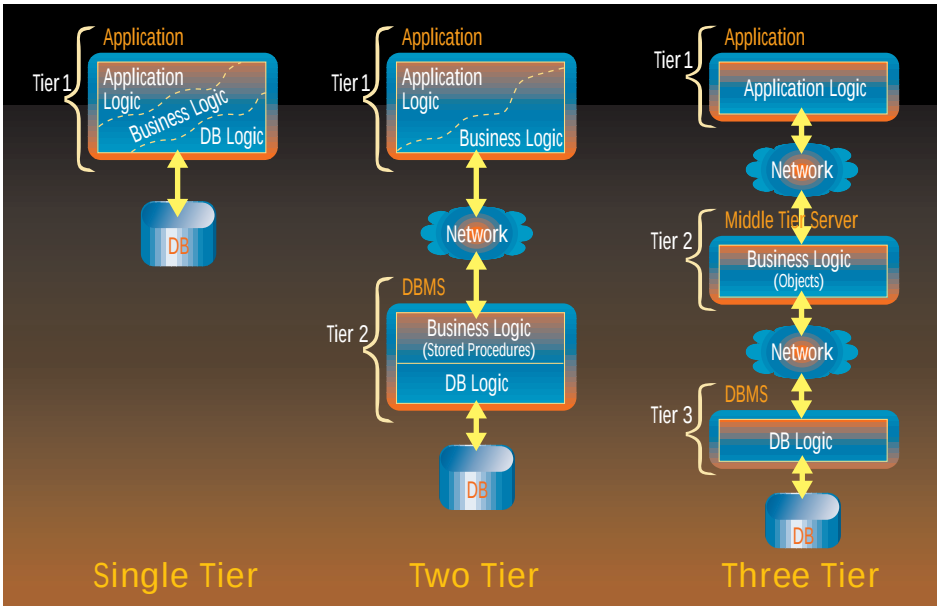


Figure 1: Application tiers

create client-side object stubs and server-side object skeletons based on the implementation classes. The stubs forward RMI calls from the client over the network to the server skeletons, which in turn forward them to the remote objects. The remote objects execute their remotely invoked methods and work gets done.

To get it all running, start the RMI Registry on the server. This basic naming service lets clients obtain references to remote objects. Each remote object must register itself with the RMI Registry when instantiated. Client applications connect to the registry to look up references to the registered remote objects. Once obtained, the reference is used to invoke the methods of the remote object. By default, RMI uses TCP/IP sockets for communication, although other transport methods can be implemented.

JDBC-compliant solutions can be networked by using the RMI--JDBC bridge. This free product comes with a Type 3 JDBC driver for use in your application. This driver connects to the RmiJdbc server that is included. The server uses any JDBC driver to connect to the database of your choice. Figure 3 shows the RMI to JDBC bridge.

Evaluating Solutions

There are many things to consider when evaluating database solutions. The criteria that are important for a specfic application depend heavily on the requirements and constraints unique to that problem. In this article, the requirement that the database is for stand-alone use with Java affects which criteria are important. The criteria most affected by this requirement are discussed below.

Database Type

The type of database used impacts development effort and application complexity. Those with no object orientation require you to map your objects to their structure, which takes time and requires code. You may also lose the ability to preserve and exploit relationships between objects. A DBMS is beneficial if it provides functionality that doesn't have to be built into the application.

To fully leverage the object-oriented nature of Java, some type of object orientation in the database is desirable. This generally means using an OODBMS or ORDBMS of some sort. Some solutions provide tools and frameworks to help map objects to relational structures and to put a more object-oriented face on relational databases. The capabilities of a DBMS of any type can be valuable as well.

Database Format

The database can be in a *proprietary* format or what I call a *standard* format. A proprietary format is just that, and is usually unique to a solution. By "standard" format I mean a common or established format that is accessible by a number of DBMSs and tools. Which option is best depends on your application.

Standard databases provide access to legacy data, use familiar file formats and have mature administration tools. They also provide a way of transitioning to Java without learning a new database. Two drawbacks include a lack of power in the native database, and having to rely on an outdated format. Another disadvantage is that your only options for accessing some common desktop databases may be to use a JDBC–ODBC driver or to roll your own solution.

Many newer solutions designed for Java use proprietary database formats. This is not necessarily a bad thing. After all, in an evolving environment today's proprietary format might be tomorrow's standard. Proprietary formats often provide significant added value, such as transaction processing, disaster recovery, object orientation and replication. One drawback of proprietary formats is the potential lack of development and administration tools. In addition, proprietary formats may still be immature and subject to the risks associated with early adoption.

JDBC Compliance

JDBC compliance provides many benefits but may not be necessary for all applications. A major advantage is the ability to make applications modular by decoupling them from a specific database solution, allowing you to plug in a different database implementation without changing the application. Another benefit is database scalability to address performance or functionality issues. JDBC also supports adding tiers for distributed deployment.

One drawback of JDBC-compliant solutions can be performance. Type 1 drivers have multiple levels of translation that inhibit performance. Likewise, Type 3 drivers use middle tiers that in turn may use Type 1 or Type 2 drivers. Another problem is deployment complexity, since there may be various libraries and executables required on the client machines or on middle-tier servers. This is true for Type 1, Type 2 and Type 3 drivers. And finally, if you need to access a common desktop database, you may not be able to find any drivers except the Type 1 ODBC to JDBC bridge.

Multiuser Implementation

Shared database access can be implemented in a number of ways with Java. Some solutions provide for multiple transactions, multithreading or both. Other solutions implement multiuser control through classic table and record locking, or feature more object-oriented methods of controlling concurrent object access. Still others require that the application implement and enforce all multiuser control.

However shared access is implemented, there's one thing most multiuser solutions for Java have in common: they require some type of server to provide network access to the shared database. This means that a multiuser Java database application will most likely have a two-tier architecture with a server component. Many solutions come complete with a server that supports shared database access, while others require that you create your own server. A two-tier architecture also requires communication between the clients and server. Sharing databases over a network requires a transport protocol. Java uses TCP/IP sockets by default with other methods supported.

How shared access is implemented significantly impacts development effort and deployment complexity. One of the biggest issues to consider is the amount of work needed to share a database. This may include writing a database or object server, developing application-level concurrency control or extending the solution using RMI. Another issue is how much work the user must do to install and administer the database. Components such as the database server, RMI registry and ODBC drivers need to be installed, configured and maintained.

Also important to look at is the difference between the single-user and multiuser versions of the solution. Some solutions offer an easy migration path while others require supporting two different versions of the application. There may also be cost differences since some solutions offer an entry-level version that supports single-user access, with a more expensive enterprise-level version required for shared access.

One risk is that a multiuser solution may not really be multiuser. Ideally, the solution should support multithreading and multiple transactions. Some solutions may emulate concurrent access but be serialized at a low level. This becomes an increasing problem as transaction rates rise.

Robustness

Robustness encompasses both resistance to failure and ease of recovery once a failure occurs. Applications typically provide some robustness but the database must be robust as well. This is especially important for stand-alone databases that need to run with little or no user administration. Ideally, failures shouldn't happen in the first place. However, given that some failures are inevitable, a robust solution should make recovery as easy as possible.

One type of common failure is data inconsistency, which results in incorrect information even though the database itself is still functional. Common causes are partial updates, improper validations and communication failures. Guarding against inconsistency may be the responsibility of the DBMS, the application or both. Solutions that support transaction processing and enforce database structure offer better protection against this type of failure.

Another kind of failure is corruption of the database files or their structure, which renders them unreadable or inaccessible. Some solutions provide low-level recovery functions for repairing corrupted databases. If the database is in an open format, other repair tools may exist. Some solutions even sport transaction logging with full automatic recovery. Depending on the solution, you may have to build some error recovery into the application to ensure adequate reliability. DBMSs generally provide increased robustness through their functionality. Be wary of solutions that are easy to corrupt through simple mistakes.

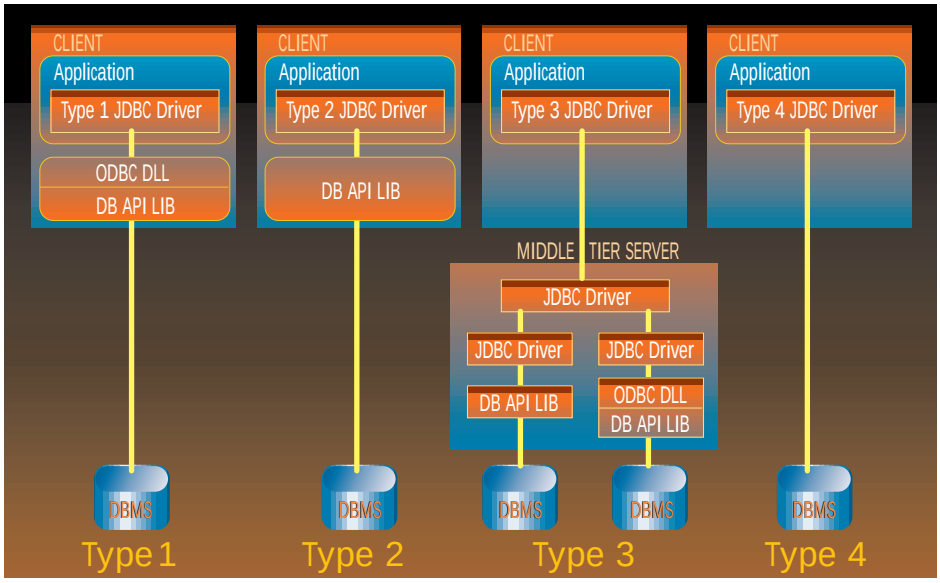


Figure 2: JDBC driver types

Portability

Since it's one of the major advantages of Java, you must consider the portability of all of the components of the solution. This includes the database server, tools, code generators, data files and any special libraries or drivers. The main requirement for portability is that the solution be 100% Java. Most solutions are 100% Java on the client side but not necessarily on the server side. Depending on the application's architecture and target platforms, the portability of any server-side components may be less important. If libraries or drivers are required on the client, they must be portable or available for all target platforms.

The risk associated with not choosing a portable solution is that your application may not run on all platforms or be able to be developed on specific platforms. The importance of this depends on the target platforms for the particular application.

Modularity

A solution's modularity can be gauged by how easy it is to implement in an existing application and, once implemented, how easy it is to remove it or substitute another solution. The easier these things are, the more modular the solution. Modularity provides flexibility, which allows unexpected issues to be addressed quickly and easily. JDBC-compliant solutions are inherently modular because of the modularity of JDBC. Solutions that are not JDBC-compliant or that present abstraction layers on top of JDBC tend to be less modular since the application code is coupled with the solution. A solution's modularity also depends on how you implement it in your application.

While modularity is desirable, other factors may be more important – for instance, using an application generator or persistence framework to implement object persistence. The solution may not be modular since the application is tightly coupled to the solution, but the productivity benefits may outweigh the potential drawbacks.

Significant Limitations

When selecting a solution, be alert for limitations that have implications for the application. Many Java database solutions suitable for stand-alone use are relatively new and not fully implemented. If limitations are found, there may be upgrade paths that provide relief. A JDBC-compliant solution, for example, lets you easily change the database solution to address limitations.

The database implementation may have limitations such as missing data types, inadequate indexing capabilities, partial SQL implementation or lack of tools. Any solution worth considering probably provides enough database functionality for most applications, but look more closely if there are special requirements. Also, look for performance limitations. Factors like application architecture, multiuser implementation and JDBC driver type all affect performance. Often performance limitations aren't evident until you subject the application to real operating conditions.

Significant Value Added

Some solutions provide significant value added for both the developer and the end user. Features like error recovery, object orientation, replication, tools, code generators and even report writers are examples of added value that come with some solutions. Since many solutions are relatively new, you should ensure that important features and

additions work as advertised. The portability of add-ons and tools may also be an issue.

Look beyond the solution vendor for other sources of added value. Standard databases often have value added from existing tools and support for the format. JDBC-compliant solutions allow use of generic JDBC-based tools.

Moving to Distributed Deployment

The application may be stand-alone now, but it might not always be so. It's worth considering how easy it is to migrate a solution to distributed deployment. Some solutions easily scale from one tier to three or more with little if any change to the application. Others may provide a migration path through a related set of products. Some solutions aren't easy to migrate beyond a one- or two-tier architecture. JDBC-compliant solutions can support multiple tiers by employing a Type 3 driver to communicate with a middle tier.

Administration Tools

To develop and support any database application, you need administration tools to create, modify, delete, query and otherwise maintain the database. The quality of these tools will affect your development and maintenance efforts. The database format (standard or proprietary) and JDBC compliance largely determine the choice of tools.

If the database uses a proprietary format, you may be dependent on the vendor for maintenance tools. An exception is if the solution is JDBC-compliant. There are an increasing number of JDBC-based tools available to maintain compliant data sources. If the database is in a standard format, tools are probably already available. Some solutions come with their own tools but functionality varies.

User Administration

User administration refers to the work the user must do to install and run the application. By definition, the user will do most of this for stand-alone applications, and it's important to make things as easy as possible. A Java application is not inherently more difficult to install and run than any other executable file. The potential for difficulty arises when it comes to the database and related components. If a solution has multiple components, any one of them may require administration by the user. Typical components requiring administration include database servers, the RMI registry, ODBC drivers, native DBMS drivers and communication protocols like TCP/IP. It's important to consider the demands that the entire solution places on the user when making your choice.

Some solutions are specifically designed for zero administration. Newer solutions are more likely to require less administration, but probably entail a proprietary database format. A true zero administration database with full automatic recovery is ideal.

Cost

Cost is always an issue when choosing a solution. This is especially true for stand-alone database applications since typically their scale is small. The total cost of the database solution must not be so high that an enterprise-scale application is required to justify it.

Most solutions require a development license, usually on a per-developer basis. Typical developer licenses cost anywhere from nothing to a few hundred dollars per license. Many solutions also require you to pay for runtime or deployment licenses. You want this cost to be low since a license is usually required for each copy of your application. Deployment licenses range from nothing or less than a dollar all the way up to several hundred dollars per client.

> ***"Multiple tiers provide a way to advantageously partition and distribute the tasks in a database application..."***

Another typical cost is for source code licenses. You may want the source code to modify or enhance a solution. Source code licenses are not always available, but when they are they can range from several hundred to several thousand dollars. Other possible costs are for any administration or support tools required to develop with the solution or to support the application.

One final cost to watch out for is some type of minimum initial investment required before you can use a product. An example would be a required purchase of a certain number of deployment licenses along with the development licenses. Although uncommon, some vendors do use this approach and the cost can be thousands of dollars.

Solution Types

Now that you know what to look for, here's a look at the various types of database solutions available for stand-alone use. Solution types are first classified by whether or not they are JDBC-compliant. Then they're roughly subdivided based on the type of database they access. Notable strengths and weaknesses of each type of solution are described below, with actual products used as examples where possible.

JDBC to ODBC Bridge

This solution is simply a Type 1 JDBC driver that converts JDBC calls to ODBC. It's an attractive solution because ODBC drivers are widely available for many standard databases and the JDBC driver is available free from JavaSoft. JDBC compliance is the other main advantage to this solution.

Drawbacks include possible slow performance due to multiple translation layers in a Type 1 driver. Deployment is complicated because ODBC is required on each client. Since most ODBC drivers aren't networked, the RMI to JDBC bridge may be required for shared access. Portability can be an issue depending on the choice of database and ODBC drivers. There are many products available in this category. Companies like Intersolv and Openlink Software sell ODBC drivers and the JDBC to ODBC bridge is available free from JavaSoft.

JDBC to Standard Database

Solutions of this type use JDBC to access a standard database format directly using a Type 2 or Type 4 JDBC driver. A solution accessing a standard database with a native API, all Java driver (not defined) would also fall into this category. Type 3 drivers don't fit in this category since they interface with a middle-tier server and not with the database. Advantages of this type of solution include JDBC compliance and the benefits of a standard database format. Performance should also be better than with the JDBC to ODBC bridge.

One major disadvantage of this solution is that Type 2 and Type 4 drivers aren't commonly available for databases suitable for stand-alone use. Deployment may be complicated for a Type 2 driver if an external library is required to access the database. Products suitable for stand-alone use aren't readily available in this category. JDBC access to traditional desktop databases is usually via the JDBC to ODBC bridge.

JDBC to Proprietary Database

This type of solution may also use a Type 2 or Type 4 JDBC driver to directly access a

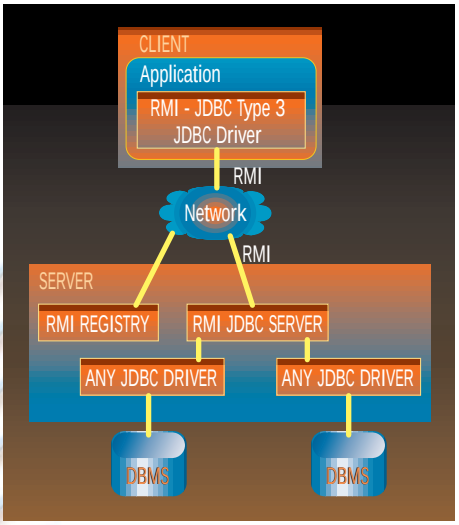


Figure 3: RMI to JDBC bridge

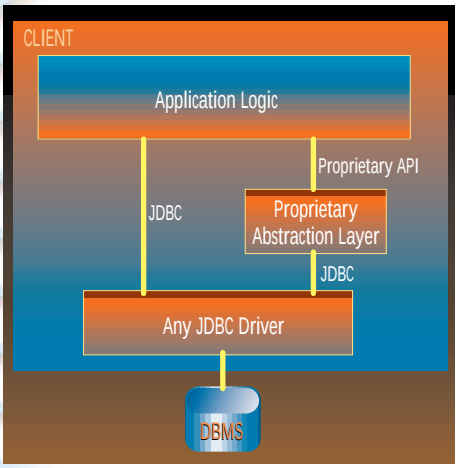


Figure 4: Abstraction layer to JDBC

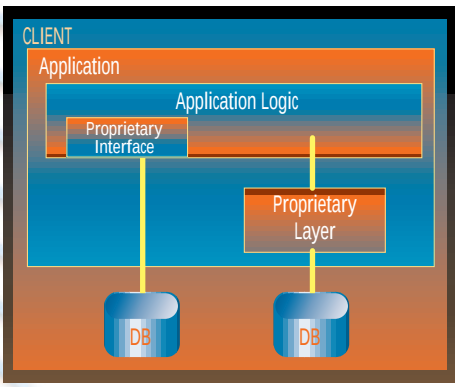


Figure 5: Non-JDBC database access

proprietary format database. As you might expect, the database vendor usually supplies the driver. Some solutions come with a JDBC-compliant driver that doesn't fit into one of the four defined types. In addition to the advantages of JDBC compliance, solutions with proprietary database formats often offer added value. Typical features include DBMS functionality, object orientation, low administration design, transaction processing and replication. Since many proprietary formats for Java are new, potential drawbacks include early adopter risks and a lack of mature support and administration tools. The flexibility of JDBC mitigates the risks associated with a proprietary solution.

One example of this type of solution is InstantDB by Instant Software Solutions, Ltd. This proprietary database comes with an unclassified JDBC driver that is 100% Java and directly accesses a local database. This solution provides added value through some SQL implementation, triggers and administration tools. Another example is JDBMS by Cloudscape, Inc. This is a full-featured ORDBMS that provides significant value added with enhanced SQL, automatic disaster recovery, transparent migration from one to *n* tiers, low administration design and advanced replication features.

Abstraction Layer to JDBC

This type of solution provides a layer of abstraction on top of JDBC in an attempt to make life easier for the developer. In some solutions this abstraction layer is designed to help bridge the gap between the object-oriented nature of Java and the relational nature of JDBC. In other solutions the abstraction layer simply provides a higher level programming interface than JDBC. Direct JDBC access is usually available if needed. These solutions often provide significant added value in the form of assistance in mapping objects to relational tables, Java and SQL code generation, and transparent migration to multiple tiers.

A potential drawback of these solutions is a lack of modularity at the application level due to the proprietary abstraction layer. JDBC compliance provides flexibility on the database end that mitigates this effect. Figure 4 shows a diagram of this type of solution.

One product in this category is CocoBase Lite from Thought, Inc., which provides the CocoBase API abstraction layer. This API implements persistence by mapping each class to one relational table. If you write your code in terms of the CocoBase API, you can upgrade transparently to CocoBase Enterprise to migrate to a multitier architecture.

Another product in this class is JDBC-Store from LPC Consulting Services, Inc. This product comes with a workbench application that helps you build a model that maps objects to a relational database schema. The workbench automatically subclasses objects and generates Java and SQL to implement transparent persistence using any JDBC data source.

A final example is DBTools.J by Rogue Wave Software, which is an abstraction layer that provides database replication and synchronization functionality. It also includes wrappers for JDBC classes that provide enhanced exception handling.

Non-JDBC to Standard Database

These types of solutions let you access a standard format database without using JDBC. Generally, this type of solution provides access only to a single database format. There aren't many of these solutions currently available since the JDBC to ODBC bridge accesses many standard databases. The lack of JDBC compliance is a drawback with this type of solution because it limits flexibility. Some potential advantages include compatibility with a standard database format, better performance than using the JDBC to ODBC bridge and a familiar data access API. Non-JDBC solutions may also provide lower level database access than JDBC, which is desirable and even required for some applications.

One example of this type of solution is XBaseJ from American Coders, Ltd., which provides classes to directly access and manipulate XBase files and indexes. Miscellaneous utilities and tools are available, as is a free multiuser server component. CodeBase also offers an ODBC driver and the JDBC-ODBC bridge to access the CodeBase server.

Non-JDBC to Proprietary Database

Solutions of this type access a proprietary database without using JDBC. These solutions are usually implemented as a set of classes and/or interfaces for storing and retrieving objects in the database. Classes for manipulating, indexing and querying the database may be provided as well. Since proprietary databases are usually designed for use with Java, they often have some object orientation.

This type of solution should have good performance due to a native driver. As stated above, proprietary databases often provide more object orientation than JDBC-compliant solutions. They also can provide lower level database access than JDBC if needed. Drawbacks include a lack of JDBC compliance and the disadvantages associated with a proprietary solution.

An example of this solution is Streamstore from Bluestream Database Software Corp. This object-persistence engine provides a simple interface that classes implement so they can be saved, retrieved and indexed. Classes for manipulating and querying the object store are also provided.

Other

One other type of solution is what I call a *data management framework*. This class of solution provides a database application framework that you can customize to create your application. Its advantage is that you can create a database application very quickly. The disadvantage is that you're completely tied to a solution that may not provide the functionality you need.

An example of this kind of solution is

MaxBase from Max Marsiglietti. This solution uses indexed ASCII files to store data. Data access and presentation are controlled by the MaxBase application, which can be customized somewhat to meet your particular needs.

Some Recommendations

By now you should be ready to go out and find the ideal stand-alone database solution to use for your Java application. But what exactly should you look for? Here are a couple of recommendations to get you started.

Shed a Tier or Two

Perhaps the most important requirement for a stand-alone database application is that it be simple to install and run. This must be true for the whole system including the application, database and supporting software. Multiple tiers make an application more complex, which can make it difficult to install and run. This is a good reason to try to limit the number of tiers in your application.

One-tier applications are nice because everything comes in one neat package. If you don't need to share your database, you may be able to use a one-tier solution. If you do, be sure to plan for migrating to more tiers in the future. Shared database access will most likely require a two-tier solution. Traditional two-tier RDBMSs such as Oracle and Sybase are not suitable for stand-alone use since they're complex to install and configure and require professional administration.

A two-tier solution destined for stand-alone use should be as simple as possible. There should be minimal software to install and configure on the client and server, and the database should need little or no administration. When selecting a two-tier solution, the simpler the better.

To JDBC or Not to JDBC?

JDBC is good because it provides modularity, portability and a standardized SQL-based API for accessing different data sources. Using JDBC mitigates some risks since you can change databases simply by changing the driver. Because of its many benefits, you should use a JDBC-compliant solution unless there are compelling reasons not to.

One reason not to use JDBC is to leverage experience with a particular database or database API. You may also want to consider other solutions if you have to use a Type 1 JDBC driver since the ODBC–JDBC bridge can be slow and requires special software on each client. Non-JDBC solutions often have some significant added value that may be important to your application. And finally, a JDBC-compliant solution may not provide enough low-level database access and control for some applications.

Get Oriented

Storing objects in a database that has no object orientation can create additional overhead during development and in the finished application. If objects and their relationships can't be stored directly in the database, then you have to map them to a form that can be stored. Object navigation and database queries will also be affected. Overcoming the mismatch between objects and your database structure can add significantly to your development tasks.

One way to reduce the effort is to use a database solution that has some object orientation. Object orientation can come in the form of an OODBMS or an ORDBMS. Other solutions allow you to store and index objects but don't provide DBMS features. Another approach is to use a tool that helps you create and maintain the object-to-database mappings. Several solutions offer frameworks or workbenches that help store objects in non-object databases.

> ***"...the solution you choose determines in part how easily you can adapt your application to future requirements."***

And Finally

Two final recommendations are to pay attention to the extras and keep an eye toward the future. By "extras" I mean several things, including goodies that come with a solution, tools required to implement the solution and everything needed to deploy the finished application. There can be a lot of these extras, and all of them can affect the development and maintainability of the application. Make sure you consider the whole picture when deciding on the best solution.

"Keeping an eye toward the future" means you should consider how your application might need to change. You may be developing a stand-alone database application now, but what if you need to change to more distributed deployment? How easy will it be to handle increased loads or changing data requirements? Thinking about these issues now will help you later because the solution you choose determines in part how easily you can adapt your application to future requirements.

The State of the Art

Now you know some things to look for in a stand-alone database solution for Java and how to pick among the products you find. But what exactly are you likely to find? Well, the available solutions are less mature than those aimed at enterprise-level databases. New products are still under development and alternatives are limited for some solution types. Current products span the range when it comes to design, functionality and quality. Product cost is usually reasonable, with full-featured solutions tending to be more expensive than less functional ones.

One area that is noticeably thin when it comes to current products is the ability to access popular desktop databases without using the JDBC to ODBC bridge. Some non-JDBC products are available for accessing XBase, but if you want to use JDBC to access another standard database, you'll probably have to use JDBC with a Type 1 driver. There's a lack of Type 2 and Type 4 drivers for desktop databases, which is unfortunate since they can provide faster access and require less configuration than Type 1 drivers. A native API, all Java JDBC driver that's 100% Java and directly accesses desktops' databases using a native API would also be nice. This type of driver would be faster, simpler and more portable than a Type 1 driver.

As I said at the start, there is both good news and bad news. The good news is that you can create real stand-alone Java database applications with the solutions available today. The bad news is that you'll face some extra difficulties and risks due to the immaturity of the product offerings. The bright side is that your choices will improve as more products emerge and existing products are refined. No single solution may meet all your needs, so weigh the benefits against the risks to pick the best one for you.

About the Author

Tim Callahan is a software developer and consultant living in Oakland, California. You can find him at his company's Web site at www.palocolorado.com or at tcallahan@palocolorado.com.

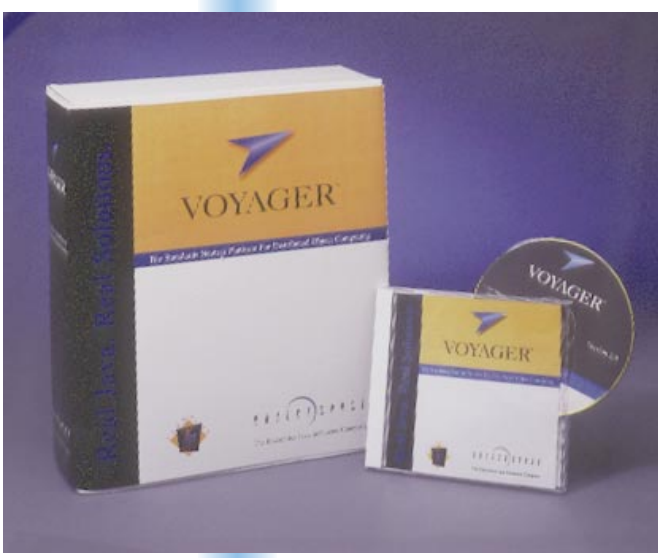

tcallahan@palocolorado.com

CASE STUDY

Accelerating Success

by **Lisa Chiranky**

Java platform hands Global Mobility a 30% time-to-market advantage



Voyager - the standards-neutral platform for distributed object computing

To gain competitive advantage, Global Mobility Systems had to speed its products to market. As the company discovered, this hinged partly on choosing the right Java platform. That is not an easy task because platforms can appear similar – until developers start using them. Global Mobility eventually selected a pure Java platform that readily placed Java's full dynamic power into developers' hands, with remarkable results.

Mike Pirie, Global Mobility's VP of product development, targeted Java early on as the language of choice for its new product development – a vehicle to deliver applications to cellular phone companies (i.e., wireless carriers). This vehicle had to manage large numbers of object-oriented components remotely across heterogeneous IT environments – a hallmark of distributed enterprise solutions.

Setting the Stage

In several industries rapid development bestows a potent advantage on nimble competitors. As a new study by International Data Corporation (IDC) indicates, Java gives development teams that edge. According to IDC, pure Java projects requiring deployment on multiple platforms yield 25% average savings compared with using a conventional OO language.

The project described in this article proves IDC's point and goes further. Global Mobility leveraged Java's power and, by choosing the right Java platform, shortened development by three months – a 30% savings on the company's anticipated 10-month time to market.

With this achievement Global Mobility is differentiating itself from competitors and reducing development time and costs. Choosing the right platform is also enabling the company to build products with more advanced features and lower maintenance costs – highly appealing to its potential customers.

Global Mobility's Market Challenge

The open system and applications developed by Global Mobility help wireless carriers bring advanced services to business users faster and more flexibly than by any other means. One application enables business cellular users to reach colleagues within a company, even when calling long distance, by dialing only the recipient's four- or five-digit in-house extension number. A second application lets cellular users contact their carrier's customer care center by dialing 611, even when roaming outside their home service area.

Advanced capabilities like these give carriers new ways to compete in a cut-throat market, enabling them to capture business customers and retain them on a basis that goes beyond commodity price per airtime minute.

Business customers make a large impact on carriers' profitability because they use the most advanced, high-margin wireless services. But with little differentiation among wireless carriers, the "churn rate" among commercial customers runs as high as 12% per month as these businesses shop

About the Author

Lisa Chiranky is director of product marketing for ObjectSpace, Inc., creator of JGL, the Voyager platform for enterprise Java solutions and C++ toolkits. For more than 20 years she has driven marketing initiatives at high technology firms ranging from startups including Landmark Graphics and Convex Computer Corporation to Fortune 500 companies including Hewlett-Packard and Texas Instruments. She can be reached at lchiranky@objectspace.com.

lchiranky@objectspace.com

Platform Strategy

Distributed enterprise solutions demand a dynamic infrastructure

by **David Norris**

Developing true distributed solutions in heterogeneous environments – and modifying these solutions in years to come – evokes traditional programming challenges in a new, more urgent context. On an enterprise scale, programming chores that could be tolerated (though certainly not welcomed) in local use become overwhelming burdens, stretching out time-to-solution, driving up costs and, by making modifications more difficult, reducing the enterprise's long-term ability to respond to business changes.

As the accompanying article demonstrates, the choice of a distributed computing platform makes an enormous impact on an IT organization's ability to create advanced applications and manage change. Organizations are discovering that ORBs alone fail to provide a sufficient infrastructure for scal-

for the best price deal. Carriers want to differentiate themselves in new ways. Service products like those from Global Mobility help them do it.

Traditionally, such products are slow to reach market. They are typically developed by large manufacturers as add-ons to proprietary telephony equipment sold to wireless carriers. Global Mobility provides carriers with service products much faster than other manufacturers do – a crucial strategic advantage both for the company and for the wireless carriers that buy its software products and services.

Global Mobility's MOE system

To realize its advantage, Global Mobility must deliver the goods in an open systems environment – unprecedented in this industry – and deliver faster than anyone else.

Technical Challenge

Global Mobility had to accelerate development of its initial applications plus a client/server vehicle for these and the company's future products. Wireless carriers use this vehicle, called Mobility Operating Environment (MOE), to integrate Global Mobility applications into their networks. At the heart of MOE lies an application manager, a unit that must perform as a distributed enterprise solution, managing remote objects across wireless carriers' disparate computing and network environments – and, in some cases, across environments owned by carriers' business customers.

Global Mobility wanted MOE to be highly portable on both the client and server sides. Furthermore, the application manager had to interact with many MOE subsystems such as protocol stacks; call processing engines; databases that store logging, tagging and provisioning information; a client-provisioning component; and a remote-alarm monitor.

The application manager also had to be able to execute on different server platforms owned by business cellular customers, part of Global Mobility's long-term deployment strategy. On the client side it had to run on Windows 95 and NT, as well as on any other box that supports a Java client, and it also had to execute on intelligent telephony devices.

That's not all. Sitting on a wireless carrier's network, MOE had to handle call processing protocols, talk to telephony switches and other components via the SS7 protocol, interact with a carrier's own TCP/IP backbone, and also reach across TCP/IP networks owned by the carrier's corporate customers in order to interact with customer databases.

All this was a tall order. Yet the technical challenge that faced Global Mobility also confronts many large IT organizations developing enterprise-scale distributed solutions for their own use.

Choosing the Right Platform

Global Mobility's development team needed robust Java facilities to interoperate with many different protocols and systems. If the company had to achieve this by modifying class descriptions using source code, it would stretch out product development. And, as Global Mobility's products evolved in a fast-changing market, it would also make it more costly and time-consuming to reuse components.

As a result, choosing the right Java platform would have a major impact on the company's ability to follow through on its business strategy. Mike Pirie took a systematic approach to the selection process.

"When you adopt a Java Platform, you are also adopting that platform's supplier as your development partner," Pirie observes. With each prospective supplier he addressed issues:

- Does the supplier and its platform have a significant track record?
- Does the supplier have a methodology certification program?
- Robust professional services?
- Education and training?

Initially Pirie explored these issues by studying brochures and Web sites, and by talking with sales reps. With this effort,

able, Java-centric distributed solutions.

Java provides a solid foundation for a scalable, distributed, standards-neutral infrastructure, but by itself is no more sufficient than a skyscraper that consists of a foundation only. Much more is needed.

Java's greatest value resides in dynamic capabilities that are far from explicit in the language. A distributed computing platform must leverage these powerful but implicit capabilities, making them readily accessible to programmers and easy to use.

The following features and capabilities help define a dynamic, pure Java infrastructure – one that reduces the time, effort and cost of building and maintaining advanced solutions across disparate systems on an enterprise scale.

Productivity

Developers become more productive when they are free to focus on adding value to their solution instead of building the infrastructure. Higher productivity provides a crucial competitive advantage, and controls development costs. A dynamic platform should:

- **Instantly remote-enable any Java class without modification**
- **Generate proxies dynamically at runtime, eliminating the need for stub generators and helper files**
- **Provide natural Java bindings, including remote pass-by value**
- **Allow remote construction of objects, eliminating the need for additional factory code**

Compatibility

A standards-neutral platform simplifies application design. It also enables developers to readily integrate enterprise and legacy systems, further raising productivity and speeding the development cycle. A properly implemented dynamic infrastructure enables objects to simultaneously interoperate with multiple ORB standards, in effect creating universal servers and clients. The infrastructure should:

- **Readily integrate systems with full native support for CORBA IDL and IIOP with bi-directional IDL/Java conversion**
- **CORBA-enable Java classes at runtime, without modification**
- **Support key RMI interfaces**

Performance

As the scale of a distributed solution increases, so does the importance of rapid, efficient, remote object messaging. A dynamic infrastructure must permit objects on geographically distributed VMs to interact without slowing enterprise applications or burdening network traffic. The infrastructure should:

- **Maximize network bandwidth with a highly optimized native protocol, minimizing code required for a roundtrip message**
- **Provide thread pooling for efficient reuse of thread resources**

Messaging

A rich feature set further improves productivity by freeing developers from the need to build the object communications infrastructure. Therefore, a dynamic platform should:

CASE STUDY

Pirie pared the prospect list.

Support was also a crucial issue. Many IT organizations, both large and small, can't find enough highly experienced Java developers. In many cases teams accelerate projects by using a platform supplier's expertise to help streamline development. Pirie sought one who was able to provide experts on short notice to work with his development team in Bellevue, Washington. Pirie held frank discussions with his prospective suppliers.

Two platforms were tested in-house. One was quickly rejected because enabling remote messaging by writing stubs and modifying classes to implement IDL interfaces rivaled the manual coding required to program in C++.

The other Java platform, Voyager Professional Edition 2.0 from ObjectSpace, eliminated the need for stubs, helper classes and source code modification. "It freed our developers to focus on their mission: delivering advanced solutions as quickly as possible," Pirie says. "That's what we were looking for."

Dynamic Infrastructure

Less than seven months after Pirie's team began developing the MOE, wireless carriers were beta-testing it along with Global Mobility's initial applications. Voyager provided the Global Mobility team with a dynamic infrastructure that sped development several ways:

- *Dynamic proxy generation* – "Dynamic proxies kept us from having to build manual stubs for Java classes," Pirie says. With other platforms, he adds, "you have to create extensive IDL and stub classes for most of the Java JDK classes. Voyager eliminated such chores by generating interface code at runtime to enable remote object messaging. This also reduced our configuration management."

  His team found that a CORBA-only ORB solution generated up to six objects (plus the associated helper classes), adding unnecessary complexity and more manual work to the solution by multiplying the number of components to be managed. Dynamic proxy generation provided in Voyager also eliminated these chores and simplified Global Mobility's solution design.
- *Distributed garbage collection* – "We had a major concern about the amount of manual programming imposed by Java platforms to clean up remote objects that were no longer referencing any particular resource within the network or our system," Pirie recalls. Voyager readily implements such distributed garbage collection. "To us, that reclaim capability was one of Voyager's most attractive features."
- *Advanced features for object communications* – Global Mobility developers found it easy to implement sophisticated functionality such as multicasting, i.e., publishing the MOE alarm service to remote clients ranging from Windows 95 PCs to intelligent telephony devices. To accomplish this, Pirie's team used Space, the ObjectSpace scalable group communication medium that came bundled with Voyager Professional Edition.

Summary

For Global Mobility Systems, choosing the right Java platform shortened time to market by 30%, reduced development costs and made major positive impacts on the feature set, performance, flexibility and ease of maintenance of the company's products.

David Norris, president and CEO of ObjectSpace

"A pure Java platform makes a lot of sense because unlike other platforms, it fulfills the promise of Java by leveraging the vast power of Java's dynamic capabilities. These enable a pure Java platform to interoperate with popular ORBs, allow dramatic reductions in manual coding, enable more advanced features in the solution and free developers to focus on their main mission – building and perfecting a solution without having to devote time and effort to building a distributed infrastructure," said David Norris, president and CEO of ObjectSpace.

As a result, Global Mobility has gained a number of strategic advantages, differentiating itself from competitors and positioning the company for future product enhancement and expansion in a fast-moving market.

Platform Strategy cont.

- **Provide an accelerated, optimized ORB**
- **Translate directly into a specific messaging protocol such as IIOP at message invocation (plus asynchronous, synchronous and one-way message delivery)**
- **Include publish/subscribe and message multicast**
- **Allow multicasting via regular Java syntax**
- **Offer distributed services such as federated directory service**

Functionality

An appropriate infrastructure transforms dynamic capabilities implicit in Java into powerful, explicit features for speeding and enhancing enterprise solutions. Examples include integrating objects with legacy databases without modifying the objects, and remotely automating garbage collection on VMs across the enterprise. A dynamic platform should:

- **Provide 100% Pure Java for "write once, run anywhere" portability**
- **Include an activation framework for database integration**
- **Allow remote class loading via an integrated HTTP server**
- **Include an integrated router, allowing applets to communicate with arbitrary servers**
- **Offer a robust security manager**
- **Support pluggable transport layers such as UDP and SSL**
- **Provide distributed garbage collection**

Flexibility

A competitive enterprise must generate a rapid, efficient response to any shift in a fast-changing business environment. Because enterprise systems support virtually all business processes, the critical success factor becomes the speed and ease with which systems can be modified. The same capabilities that enable a dynamic Java infrastructure to shorten time-to-market also shorten the "time-to-change," making the entire enterprise more flexible in responding to business and technology changes. To accomplish this, the platform should:

- **Support third-party messaging protocols and multiple transport layers**
- **Provide pluggable resource loaders for flexible remote class loading**
- **Dynamically support multiple distributed object models**
- **Readily scale to enterprise applications and embedded solutions**

Leading Edge

Extending an object's behavior at runtime opens the door to innovations that make distributed solutions even more robust – ultimately delivering greater value to users of applications across the enterprise. For example, a robust platform should:

- **Create mobile autonomous agents – objects that continue to execute as they move themselves between VMs**
- **Allow an object's functionality to be dynamically aggregated, i.e., extended at runtime (complements inheritance.)**

VoyagerPro 2.0
by ObjectSpace

A 100% Pure Java development platform and ORB for distributed computing

by **Jim Milbery**

VoyagerPro 2.0
ObjectSpace, Inc.
14850 Quorum Drive, Suite 500
Dallas, TX 75240
Phone: 800 OBJECT-1
Web: www.objectspace.com
Minimum Requirements: JVM with support for JDK 1.1 (Windows 95/NT, Macintosh, UNIX)

ObjectSpace has made a name for themselves in distributed computing over the past few years. They recently announced that Sun Microsystems had licensed ObjectSpace's JGL technology (a toolkit for building and managing collections) for integration with Sun's own JavaBlend technology. ObjectSpace has followed up with the release of their VoyagerPro 2.0 product, and I had the opportunity to work with a prerelease of the software.

Product Installation

The product ships on CD-ROM and is also available for download from ObjectSpace's Web site in a variety of formats. The CD-ROM version installs via InstallShield, and I had it installed under Windows NT and configured in a matter of minutes. A complete installation takes a little over 4 MB of disk space and doesn't require a system reboot. You need version 1.1 of the JDK on your machine to work with VoyagerPro. The software, written in Java, will run on any platform that supports a JDK 1.1 VM.

VoyagerPro Features

The VoyagerPro product, which is aimed at the professional developer marketplace, comes with a number of features aimed at providing distributed computing capabilities for your Java applications. The folks at ObjectSpace have bundled a lot of functionality into this product, and it's impossible to adequately cover all its capabilities here.

One impressive feature is a technology that ObjectSpace calls *Dynamic Aggregation*. It's particularly powerful in that it allows the developer to add functionality to third-party components, even if you don't have the source code for that component. Dynamic Aggregation goes beyond what inheritance and polymorphism provide for the object developer, as it allows the developer to attach secondary objects (which ObjectSpace refers to as *facets)* to a primary object at runtime. The primary class doesn't have to be related in any way to the facet class, and you don't have to modify the class files of either object. VoyagerPro includes additional facilities for remote-enabling classes, synchronous and asynchronous messaging, CORBA translation and even mobile agents that can move themselves between programs.

VoyagerPro also features nice facilities for building multicast and publish/subscribe applications. It allows you to specify distributed containers, called *Spaces*, that can span programs. Multicast messages can be propagated automatically between the subspaces within a space, and VoyagerPro detects when duplicate messages are received.

The ObjectSpace team assumes that VoyagerPro will be used by experienced Java developers, so the examples are almost always short, direct and to the point. ObjectSpace includes sample code for all of the product's features, including Dynamic Aggregation. I built and ran the example aggregation code for employees and accounts without difficulty. I find it much easier to work with this type of example code.

VoyagerPro Components

VoyagerPro is packaged with a set of four utility programs and a Java archive of the VoyagerPro classes. The IGEN utility is used to generate an interface for your specified class file. You're allowed to specify the Java intepreter that will be called by IGEN, and the result of running IGEN will include all of the public methods for your specified class hierarchy. The CGEN, or code generator, generates IDL from Java files and Java from IDL files, allowing you easy access to CORBA from the Java language. Although VoyagerPro supports dynamic proxy generation, it also includes the PGEN utility for manual proxy generation when performance requirements or postprocessing needs justify the use of manual proxies. The main utility is the Voyager program itself, with the VoyagerPro development server and ORB. The Voyager server uses a small footprint and includes its own HTTP listener for serving classes to other remote Voyager servers. Voyager is completely managed from the system command line, and any settings or parameters you wish to supply are passed on to the command line in a format familiar to UNIX programmers. The combination of the Voyager server and the VoyagerPro classes makes it easy to deploy a multitier application with a minimum of clutter. Although the Voyager server supports many common application server features such as logging, class loading at startup, listener port and Java intepreter parameters, the server itself offers a bare-bones interface. I was impressed with how quickly I could remote-enable VoyagerPro's example code.

VoyagerPro also includes an activation framework that allows objects to be persisted into a database without having to modify the object's class. I'd recommend VoyagerPro to developers who are looking to add powerful functionality to their Java applications using a simple, standards-neutral interface. Bookmark ObjectSpace's Web site, as they have recently announced some VoyagerPro add-ons in the form of transaction services, security services and Enterprise JavaBeans.

About the Author

Jim Milbery, an independent software consultant based in Easton, Pennsylvania, has over 15 years of experience in application development and relational databases. You can reach him at jmilbery@milbery.com or via his Web site at www.milbery.com.

www.milbery.com

Focus: The Java Scripts

They're easy to use...and what's more, they're free!

by **Ajit Sagar**

The world of software programming is replete with alternative tools for writing code that can be used to provide the same solutions to the same problems. The range of programming aids available, and their disparate approaches, make ubiquity and platform neutrality a myth. Nowadays, someone who says that the Z80 assembly language is also a programming language will be shot down with a barrage of e-mail arrows fired by technical gurus. I think the only absolutely platform-independent, language-neutral, ubiquitous truth is that everything ultimately translates into a sequence of zeros and ones.

In the world of programming languages, the kingdom of "high-level programming" is shared by two types of languages: *system programming languages* and *scripting languages*. High-level languages such as C++, Java, Smalltalk and Pascal basically abstract programmers from the low-level, machine-specific programming details required in assembly programming. System programming languages and assembly languages have the same purpose: to develop applications directly using system resources. Scripting languages on the other hand supplement system programming languages by providing "glue-code" to integrate components written in other languages (usually system programming languages). Applications and components written in system programming languages can be glued together for quick prototyping, configuration and deployment using scripting languages.

The Cosmic Cup's purpose is to allow true believers to peer into the different facets and pieces of the Java Platform. The platform would be incomplete without scripting language support. As the Java programming language gained popularity, two kinds of evolutions took place in the world of scripting languages. The first involved the inception of new languages that facilitated the integration of Java into the Web – more specifically into HTML. The other involved the enhancement of existing scripting languages that added Java support to their existing codebase. These scripting languages differ from the former in that they have a substantial presence outside the Internet. The languages in this category enhanced their features and provided hooks to integrate components from non-Java environments to components developed in Java.

This month we'll take a look at the Java support provided by five popular scripting languages:

- Tcl
- Python
- JavaScript
- JScript
- VBScript

Of these, the first two provide enhancements to existing languages; the other three are primarily HTML-based scripting languages. Our discussion will focus primarily on the three languages in the first category.

The role of these languages in Java-based development is illustrated in Figure 1, and brief descriptions are provided in Table 1. The remainder of this column describes these scripting languages in more detail.

Tcl/Tk

Tcl, built on the C environment, is an ideal scripting language for embedding into other applications. It focuses on small scripts, rapid application development and dynamic environments. Started in 1990 as a research project at the University of California, Berkeley, by John Ousterhout, it has been supported informally by Sun since 1994. It is available on popular computing hardware platforms (UNIX, Windows, Macintosh) and is used by about half a million programmers.

A salient aspect of Tcl is that it can be treated as a library and easily embedded in existing Java applications. Java developers can use this functionality as a means to wrap existing legacy components into their Java applications. It can also be used as the glue that ties together components developed in Java, adding flexibility, dynamism and rapid integration to the development process.

Tk is Tcl's toolkit of widgets, graphical objects similar to those of other GUI toolkits such as Xlib, Xview and Motif. Tk can be used to create prototype GUIs with the Tcl scripting language.

The Tcl-Java integration effort involves development in the following areas that

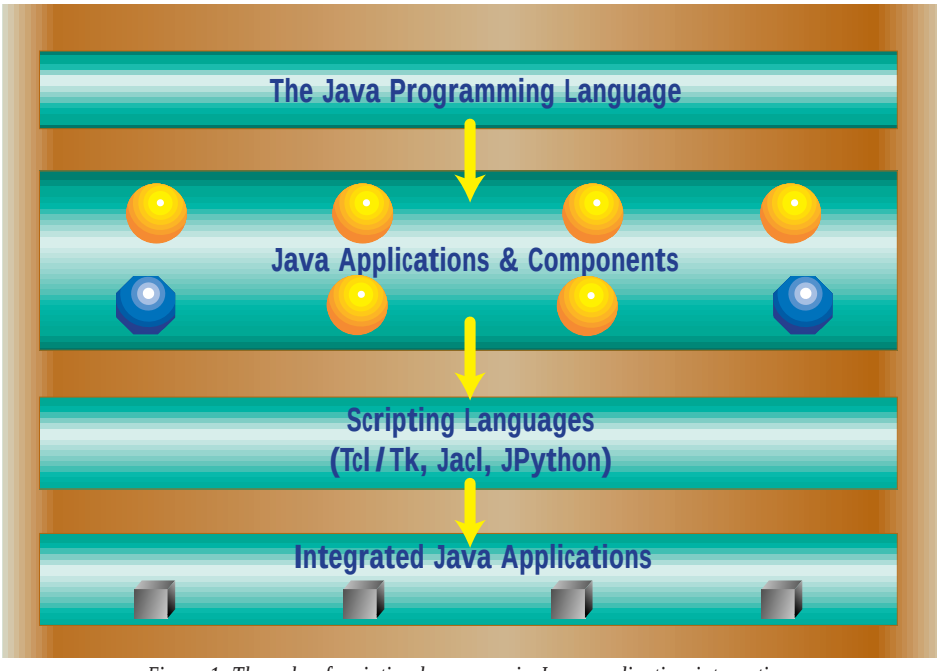


Figure 1: The role of scripting languages in Java application integration

add functionality to different facets of the Java Platform:

- *TclBlend* – A new package for Tcl that allows Tcl applications to load and interact with the Java Virtual Machine. This allows developers to use Java objects in their existing legacy code. The final 1.0 release is available for Solaris and NT and uses Tcl 8.0 with JDK 1.1.
- *Jacl* – Tcl's 100% Java solution, the objective and result of Tcl's port to the JVM. The Jacl effort involves porting the entire Tcl and Tk code to the Java programming language, thus providing a Pure Java solution that includes a system programming language with tightly integrated scripting support. Currently, the core of the Tcl language is being ported. The Tk widgets are being wrapped into JavaBeans to reuse existing code. (Jacl doesn't yet have applet support, i.e., it doesn't run in browsers.)
- *Tcl Bean* – A JavaBean built for Java Studio – Sun Microsystems' Java IDE – is used to create ports and generate messages via the studio package. Java Studio uses ports to graphically represent JavaBean events.

Sun recently created a new business unit, Sunscript, which has evolved into a company called Scriptics, the official owner of Tcl/Tk. Detailed information on the Tcl-Java integration is available at www.scriptics.com/java.

Python

Python is a portable, interpreted, object-oriented programming language that incorporates modules, exceptions, dynamic typing, high-level dynamic data types and classes. Created in 1990 by Guido van Rossum, it's currently distributed freely, and is maintained by an informal development organization called the Python Software Activity, or PSA, which houses a large developer community. Python's syntax is closer to traditional programming languages such as C. It may be used for rapid prototyping as well as for medium- to large-scale systems development.

JPython is a new implementation of Python integrated with the Java Platform. Recently certified as 100% Pure Java by KeyLabs, it consists of a compiler that translates Python source code into Java bytecodes that can run directly on a JVM, a set of support libraries used by the compiled Java bytecodes and extra support for using Java packages from within JPython.

JPython enhances the functionality of Java programs by providing programmers with a rapid application development environment for integrating Java components. Java programmers can add the JPython libraries to their system to allow end users to write simple or complicated scripts that add functionality to an application. JPython has an interactive interpreter that allows dynamic interaction with Java packages or Java applications.

As JPython is written in Java, interoperability between the two languages is more intuitive. JPython provides the following functionality to the Java programming language:

- *Dynamic compilation of "glue-code" written in JPython to Java bytecodes.* This makes the executable code available to all platforms that support the JVM.
- *Inheritance capability that allows JPython code to extend Java classes.* Thus implementation of abstract classes defined in a Java program may be provided by the scripting code.
- *Support for the JavaBeans component model.* JPython uses JavaBean properties that make it easier to interact with most Java classes. These properties can be used as normal object attributes, and can also be specified to the class constructor as keyword arguments. JPython uses Java's Introspection mechanism to achieve this.

Detailed information on JPython is available at www.python.org/jpython.

Scripting Language	Description	Type
Tcl	A scripting language ideal for embedding into other applications that focuses on small scripts, rapid application dvelopment and dynamic environments	Independant scripting applications
Python	A portable, interpreted, object-oriented scripting language that incoporates modules, exceptions, dynamic typing, high-level dynamic data types and classes	Independant scripting applications
JavaScript	An object-based interpreted scripting language that is embedded in HTML pages	HTML support
JScript	Microsoft's implementation of JavaScript that adds support specific to Internet Explorer	HTML support
VBScript	A subset of the Microsoft Visual basic programming language that offers a lightweight interpreter for use in World Wide Web browsers	HTML support

Table 1: Scripting languages

Web Scripting Languages

The remainder of this article focuses on the three HTML-based scripting languages listed earlier. The sole purpose of these languages is to enhance the content of Web pages displayed by Web browsers. In the current Internet world "Web pages" inherently implies HTML authoring. The languages discussed below allow the programmer to add sophisticated Web content to Web pages, and to support object orientation and the capability of embedding programming language components into Web pages. In other words, the Web scripting languages provide "glue-code" for Internet (or intranet) Web pages. This aspect makes the scripting language dependent on the Web browser that displays the Web page. The browser should be able to recognize the tags for a particular language. If it doesn't, all code between the open and close tags (the code provided for the respective scripting language) will be ignored.

The scripting language code is embedded in the HTML page as follows:

```
<HTML>
.....
 <SCRIPT LANGUAGE = "[language]" >
  [Scripting Code]
 </SCRIPT>
.....
</HTML>
```

The "[language]" string could be "JavaScript," "VBScript," "JScript," etc.

I'll provide very brief descriptions of these scripting languages as there are ample sources of information that discuss them in detail.

JavaScript

JavaScript provides object-based interpreted scripting that is embedded in HTML pages. Originally developed by Netscape as "LiveScript," Netscape and Sun Microsystems entered into an agreement to jointly enhance the language in December 1995.

The name was changed to JavaScript as the addition of Java support was the main aspect of the enhancement.

JavaScript syntax is similar to that of the Java programming language. It also uses elements from the Awk and Perl scripting languages, and its object-based nature can be attributed to the object prototype system used in the Self language. JavaScript is supported by Netscape Navigator versions 2.0+ and Microsoft Internet Explorer.

JavaScript also has a server-side component, LiveWire, which contains server-side Java code and extensions to the client-side JavaScript. This code is compiled into Java bytecodes and can be used as an alternative to CGI. Since the 1.2 release of JavaScript, the client- and server-side versions have been consolidated into a single technology supported by Netscape Communicator 4.0 and Netscape's Enterprise Server.

Detailed information on JavaScript is available at http://developer.netscape.com/openstudio/tech/index_frame.html.

JScript

JScript, Microsoft's implementation of JavaScript, adds support specific to its Internet Explorer browser. Like JavaScript, JScript allows developers to link and automate objects in Web pages, including ActiveX controls and Java Applets. JScript is supported in both Netscape and Microsoft browsers. Some of its key features are dynamic redefinition of the executing program, object-based support, DHTML support, a rich support for regular expressions and the capability to immediately evaluate code at runtime.

More information on JScript is available at www.microsoft.com/workshop/languages/clinic/vbsvjs.asp.

VBScript

A subset of Microsoft's Visual Basic programming language, VBScript offers a lightweight interpreter for use in World Wide Web browsers, and other applications such as ActiveX controls, Automation servers and Java Applets. The main features of VBScript are:

- It adds Web development capabilities to the client and server.
- It brings a useful scripting language to the Web.
- It expands the scope of the Visual Basic programming languages to platforms not covered by either Visual Basic or Visual Basic for Applications.

More information on JavaScript is available at www.microsoft.com/workshop/languages/clinic/vbsvjs.asp.

Cosmic Reflections

System programming languages and scripting languages try to address application development in their own way. The scopes of programming covered by these languages usually overlap. Though they supplement each other, they also compete with each other. The main attraction of scripting languages is their ease of use and their extremely rapid development. Furthermore, they're free. Developers need to carefully consider the roles and division of responsibilities they want to assign to the chosen scripting language when they are architecting enterprise-level applications. Recent developments and the direction of the popular scripting languages seem to indicate that the Java Platform is not just Java.

About the Author

Ajit Sagar is a member of the technical staff at i2 Technologies in Dallas, Texas. He holds a BS in electrical engineering from BITS Pilani, India, and an MS in computer science from Mississippi State University. He is a Java certified programmer with eight years of programming experience, including two in Java. Ajit can be reached at Ajit_Sagar@i2.com.

Ajit_Sagar@i2.com

Dynamic Java Debug Code

It may be a little slow, but this new technique means less development overhead and better coding style

by **Joe Chou**

Object-oriented developers who move from C++ to Java miss the preprocessor capability of C++. That capability allows them to include and exclude debug code easily simply by changing the compiler switches in the development and release versions of the application. But Java's lack of preprocessor capability doesn't mean you can't incorporate debug code in your Java programs. Some developers have come up with the idea of using debug and nondebug packages conditionally to mimic the conditional compilation available in the C++ preprocessor.

The downside of this approach is that you have to implement the same debug class twice: once for development, then again for release. The development version of the debug class will have debug statements implemented in every method; the statements will be removed in the release version. The implication of maintaining the same class with different implementation in two places is that managing changes in the debug class can be tedious and error-prone. Even worse, every package using the debug class will have to be modified to include the corresponding package – a daunting task in itself.

This article presents a cleaner solution for inserting the debug code into your Java applications by taking advantage of Java's dynamic class loading during development. Using this technique, debug code is written only once. Also, because there's only one version of the code, using out-of-sync debug classes is no longer a problem. Moreover, the requirement for changing the import statements for including the correct debug packages before releasing the program is completely eliminated, which saves significant development time by avoiding retesting.

Note: Although the technique can be applied to both Java 1.0 and 1.1 versions, the examples in this article utilize the Class Literals introduced in Java 1.1, which makes the examples simpler and safer than if they were written in Java 1.0.

Implementing Java Debug Code

Most Java developers with a C++ background have a hard time learning not to use the C++ preprocessor when writing Java applications. You might have heard that in Java you can obtain the effects of most C++ preprocessor features by using Java language elements, i.e., constants in place of #defines. You declare classes in place of #typedefs. Because Java supports platform portability natively, you don't need conditional compilation.

Unfortunately, although Java designers have managed to leave out most of the unnecessarily complicated language elements in C++, the removal of conditional compilation also eliminated a common and valuable use it provided – debug code. Thus the need for ways to place and remove debug code in the development and release versions of Java applications.

The conditional debug packages technique requires substantial effort to fix up the code before shipping the Java application. The new technique, which uses Java's dynamic class loading, requires no effort. To illustrate and compare the two techniques, we'll show you the Java version of C++ assertion using both techniques. First, let's see how the conditional debug packages technique works.

Conditional Debug Packages Technique

This technique requires two classes with identical class names and interfaces: one used during development, the other for

release. The only differences between the two classes are the packages the classes are in and the classes' method implementation. In my example the development class is located in the debug package and the release class is stored in the tools package. You're free to name the two packages, of course, as long as the names are meaningful and short. You'll implement the methods of the development class but keep the methods in the release class empty.

Now we're ready to implement the Java version of C++ assertion using the conditional debug packages technique. We'll name the debug class CondDebug. For efficiency we'll define the class as final since you won't normally want to extend a class that'll be used only for debugging code. For ease of use we define the two overloaded assert methods as static and disallow the instantiation of CondDebug class by making its constructor private.

Listing 1 shows the development version of CondDebug class. A CondAssertException class is thrown to stop the application if the condition passed in the assert method is false. Listing 2 shows the CondAssertException class. If you look at the code closely, you'll notice that CondAssertException class is a subclass of RuntimeException instead of Exception. There are two reasons for this: (1) the CondAssertException will not be caught in the code because it will be used strictly to stop the execution of the program, and (2) you don't want to declare a throws clause for every method calling the assert method. Since the CondAssertException class doesn't require enforcing Exceptions, it makes sense for the CondAssertException to be a subclass of RuntimeException.

Listing 3 shows the release version of the CondDebug class included in the tools package. As you can see, the only difference between this class and the one in the debug package is that assert methods in this class are empty. This way, the release version of the application will simply pass through the assert method, causing little performance penalty in the overall program execution. We'll examine the performance issue later in this article.

Listing 4 shows an example, TestCondDebug class, of how you can use the different versions of the CondDebug class by manually modifying the package being imported. Note that the main method catches both CondAssertExceptions so the first CondAssertException won't exit the program. This way, the second assert method can be tested without rerunning TestCondDebug. This will be the only case where CondAssertException is caught when testing CondDebug and CondAssertException itself.

The conditional debug packages technique has three disadvantages:

1. Developers will have to manually maintain two copies of the debug classes that can't be checked with the help of the Java compiler.
2. Import statements in classes that use the debug classes will have to be changed before the code is ready for shipping. This poses no problem for a small Java program, but for a large Java program that consists of hundreds of classes, a script will have to be written to automate the process. This script will then have to be maintained to keep up with new debug classes introduced during development.
3. Because of the amount of code changes required at the end of product development, a full regression test must be performed to ensure that the release version of the application contains no new defects introduced by the code changes. This test, performed at the last minute of a product cycle, is simply unnecessary and costly.

Dynamically Loading Debug Classes Technique

In the section above you read about the conditional debug packages technique and learned about its drawbacks. So the question is: How do we remove the tedious and error-prone code changes and still have access to the debug code only when we need it? The answer lies in one of the powerful Java features – dynamic class loading.

With Java's dynamic class loading we can determine when to load the debug class at runtime. Because we can load the class whenever we wish, we can load it only when the program is run in a development environment. The program will know whether it is run in a development or release environment by checking for the presence of the debug class, which is removed in the release version of the application. By encapsulating debug methods in a debug class and applying Java's dynamic class loading, we can now load the debug class and execute its methods once the class is found. Only when the debug class is successfully loaded from CLASSPATH will its methods be executed. Consequently, simply adding and removing the debug class from the Java packages controls the use of the methods in the debug class. With this in mind, I'll show you how to implement the technique of dynamically loading debug classes.

Although you can call a debug class method from anywhere in your program, you might find it easier to manage if you add a layer between the calling class and the debug class itself. In other words, adding a wrapper class that transparently

encloses the debug class will consolidate all possible changes in one place and minimize the impact of the code changes. In the assert example we'll call the wrapper class JDebug (Listing 5) and include it in the tools package. We'll call the debug class Debug (Listing 6) and include it in the debug package.

JDebug class will have the same interface as the Debug class, with the addition of debugLoaded method. This method will be called in JDebug's debug method to determine whether to call the corresponding method in Debug class, based on the return value of debugLoaded method. You'll have to include debugLoaded method in every wrapper class if you declare the wrapper class final, as in the example, which will speed up the execution time of the program. If you want to take advantage of Java's inheritance and don't mind having a small performance penalty in your application, you could create a base debug class containing this protected debugLoaded method. This would allow you to extend the base debug class without worrying about copying and pasting the same method over to your new wrapper class. This decision can be made on a project-by-project basis.

In JDebug's debugLoaded method we don't handle any of the exceptions that are thrown if Debug class can't be found. This way nothing will show up in the release version of the application. However, we do print out a message (*** Debug On ***) when the code runs during development to indicate the current status. With this message, test engineers can write a simple test to ensure that the release image of the application does not contain the unwanted Debug class. If you compare the testing effort of the new technique with the full regression required by the conditional debug packages technique, you can see how much development time can be saved.

Now that you've learned how the dynamically loading debug classes technique works, it's easy to apply it to other debug methods you might want to add. As an example, I've included three new methods, printStackTrace, memoryInfo and printTimeDff, in both JDebug and Debug classes. As the code explains what they do, I won't go into detail here. You can look at JDebug's main method to get some idea of how they can be applied to your programs.

The dynamically loading debug classes technique eliminates the three disadvantages of the conditional debug packages technique and instead offers the following advantages:

1. You don't have to manually maintain two copies of the debug classes.
2. You can switch from development to production simply by removing Debug class from CLASSPATH.
3. You don't have to change a single line of code to release the application because the debug classes are dynamically loaded when they exist.
4. You don't have to retest your code, saving significant delay in the application's time to market, because there are no code changes.

Only one disadvantage is associated with this technique: the extra conditional code that checks for the existence of Debug class will have to be included in the release version of the application, which will slightly affect its performance. How much slower will the application actually be?

Performance Issue

Because we add the code to load Debug class the first time the class's debug method is called, as well as to test the existence of Debug in every method in JDebug class, it's evident that some performance penalties might be associated with the additional code. We've written ProfileCondDebug and ProfileDynamicDebug classes to measure exactly how much time is required to run the additional code. In both classes we measure the time needed to call the assert method 10 million times: 109 ms for the CondDebug.assert method and 1,109 ms for the Debug.assert method on a Pentium 200 MHz machine running JDK 1.1.6 on Windows NT 4.0.

Even though the Debug.assert method runs about 10 times more slowly than the CondDebug.assert method, each Debug.assert method still takes only 0.11 µs. In other words, you can afford to execute Debug.assert method 10 thousand times and it will slow down the particular operation for only 1.1 ms in the machine setup we have, which should be more than acceptable in most Java applications.

In a time-critical application, however, and especially in a routine that will be used many times in the application, adding debug code that will slow down the application even a little bit isn't acceptable. Does that mean you can't use this technique in such applications? No. There are still ways to apply debug code in such a situation. Before you release your time-critical application, run it with a profiling tool and identify the routines that take a large share of the runtime. Remove the debug code completely from those routines and you're done. If this still doesn't satisfy your need for speed, then you might have used the wrong language in developing your application. In this case C++, or maybe even Assembly, would probably be a better choice than Java.

Listing 7 lists ProfileCondDebug class and Listing 8 shows ProfileDynamicDebug class.

Conclusion

The dynamically loading debug classes technique is slower than the conditional debug packages technique, but as we've demonstrated, the new technique offers less development overhead than the conditional debug packages technique – yet provides much better coding style in terms of both maintainability and readability. If your application's performance is very important to its user, you might consider removing all your debug code from time-critical routines before the application is shipped to customers.

This article shows only one way to exploit Java's capability to load classes dynamically. I believe you'll come up with ideas even better than mine. Don't forget to let us know!

CODE LISTING

The complete code listing for this article can be located at **www.JavaDevelopersJournal.com**

About the Author

Joe Chou has been testing and developing software in Silicon Valley for nine years. Joe first studied C++, later worked with Delphi and now enjoys writing applications in Java at IBM in San Jose, California. He is always searching for creative design and practical techniques in software development. You can reach him at joechou@compuserve.com.

joechou@compuserve.com

Java Developer's Journal

www.JavaDeveloper'sJournal.com

Programming with Java's I/O Streams – *Part 1*

Learn the basic concepts here so you can start programming on your own

by **Anil Hemrajani**

Most programs use data in one form or another – as input, output or both. The sources of input and output can vary from a local file to a socket on the network, a database, in memory or another program. Even the type of data can vary from objects and characters to multimedia and more.

The APIs Java provides for reading and writing streams of data have been part of the core Java Development Kit since version 1.0, but they're often overshadowed by the better known JavaBeans, JFC, RMI, JDBC and others. However, input and output streams are the backbone of the Java APIs, and understanding them is not only crucial but can also make programming with them a lot of fun.

In this article we'll cover the fundamentals of I/O streams by looking at the various stream classes and covering the concept of stream chaining. Next month we'll look at some example uses of I/O streams.

Overview

To bring data into a program, a Java program opens a stream to a data source – such as a file or remote socket – and reads the information serially. On the flip side a program can open a stream to a data source and write to it in serial fashion. Whether you're reading from a file or from a socket, the concept of serially reading from, and writing to, different data sources is the same. For that very reason, once you understand the top-level classes (java.io.Reader, java.io.Writer), the remaining classes are a breeze to work with.

Character versus Byte Streams

Prior to JDK 1.1, the input and output classes (mostly found in the java.io package) supported only 8-bit "byte" streams. In JDK 1.1 the concept of 16-bit Unicode "character" streams was introduced. While the byte streams were supported via the java.io.InputStream and java.io.OutputStream classes and their subclasses, character streams are implemented by the java.io.Reader and java.io.Writer classes and their subclasses.

Most of the functionality available for byte streams is also provided for character streams. The methods for character streams generally accept parameters of data type "char" parameters, while "byte" streams – you guessed it – work with "byte" data types. The names of the methods in both sets of classes are almost identical except for the suffix; that is, character-stream classes end with the suffix Reader or Writer and byte-stream classes end with the suffix InputStream and OutputStream. For example, to read files using character streams, you'd use the java.io.FileReader class; to read using byte streams you'd use java.io.FileInputStream.

Unless you're working with binary data such as image and sound files, you should use readers and writers to read and write information for the following three reasons:

1. They can handle any character in the Unicode character set (the byte streams are limited to ISO-Latin-18-bit bytes).
2. Programs that use character streams are easier to internationalize because they're not dependent upon a specific character encoding.
3. Character streams use buffering techniques internally and are therefore potentially much more efficient than byte streams.

To bridge the gap between the byte and character-stream classes, Java provides the java.io.InputStreamReader and java.io.OutputStreamWriter classes. The only purpose of these classes is to convert byte data into character-based data according to a specified (or the platform default) encoding. For example, the static data member "in" in the "System" class is essentially a handle to the Standard Input (stdin) device. If you wanted to "wrap" this inside the java.io.BufferedReader class that works with character streams, you'd use InputStreamReader class as follows:

```
BufferedReader in = new BufferedReader(new
InputStreamReader(System.in));
```

For JDK 1.0 Versions

If you can't use JDK 1.1, perhaps because you're developing applets for older browsers, simply use the byte-stream versions, which work just as well. Although I haven't discussed these versions much, they work almost identically to the character versions from a developer's perspective except, of course, the reader/writers accept character data types versus byte data types.

The Various Stream Classes

Top-Level Classes: java.io.Reader and java.io.Writer

Reader and Writer are the abstract parent classes for character stream-based classes in the java.io package. As discussed above, Reader classes are used to read 16-bit character streams and Writer classes are used to write to 16-bit character streams. The methods for reading and writing to streams found in these and their descendant classes (discussed in the next section) are:

```
int read()
int read(char cbuf[])
int read(char cbuf[], int offset, int length)
int write(int c)
int write(char cbuf[])
int write(char cbuf[], int offset, int length)
```

Listing 1 demonstrates how the read and write methods can be used. The program is similar to the MS-DOS type and Unix cat commands, that is, it displays the contents of a file. The following code fragment from Listing 1 opens the input and output streams:

```
FileReader  fr = new FileReader(args[0]);
PrintWriter pw = new PrintWriter(System.out,
true);
```

The program then reads the input file and displays its contents till it hits an end of file condition (-1), as shown here:

```
while ((read = fr.read(c)) != -1)
pw.write(c, 0, read);
```

I used the "(char cbuf[])" version of the read method, because reading a single character at a time can be approximately five times slower than reading chunks (array) at a time.

Other notable methods in the top level classes include skip(int), mark(int), reset(), available(), ready() and flush().

- skip(), as the name implies, allows you to skip over characters.
- mark() and reset() provide a bookmarking feature that allows you to read ahead in a stream to inspect the upcoming data but not necessarily process it. Not all streams support "marking". To determine whether a stream supports it, use the markSupported() method.
- InputStream.available() tells you how many bytes are available to be read before the next read() will block. Reader.ready() is similar to the available() method, except it does not indicate how many characters are available.
- The flush() method simply writes out any buffered characters (or bytes) to the destination (e.g., file, socket).

Specialized Descendant Stream Classes

Several specialized stream classes subclass from the Reader and Writer classes to provide additional functionality. For example, the BufferedReader provides not only buffered reading for efficiency but also methods such as "readLine()" to read a line of input.

The class hierarchy shown in Listing 4 portrays a few of the specialized classes found in the java.io package. This hierarchy merely demonstrates how stream classes extend their parent classes (e.g.. LineNumberReader) to add more specialized functionality. Tables 1, 2 and 3 provide a more comprehensive list of the various descendant classes found in the java.io and other packages, along with a brief description for each class. These descendant classes are divided into two categories: those that read from or write to "data sinks&quot, and those that perform some sort of processing on the data (this distinction is merely to group the classes into two logical sections; you don't have to know one way or the other when using them).

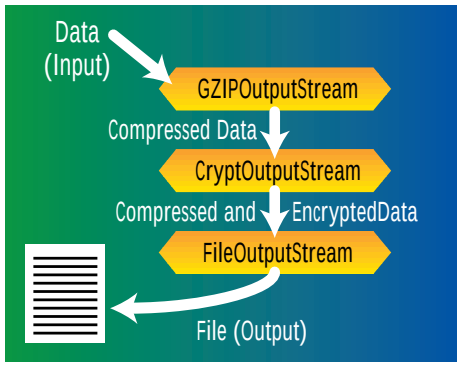


Figure 1: Stream chaining

Listings 2 and 3 don't contain the complete list for the table because I intentionally skipped the "byte" counterparts to the "char" based classes and a few others (please refer to the JDK API reference guide for a complete list).

Stream Chaining

One of the best features of the I/O stream classes is that they're designed to work together via stream chaining.

Stream chaining is the concept of "connecting" several stream classes together to get the data in the form required. Each class performs a specific task on the data and forwards it to the next class in the chain. Stream chaining can be very handy. For example, in our own 100% Pure Java backup software, BackOnline, we chain several stream classes to compress, encrypt, transmit, receive and finally store the data in a remote file.

Figure 1 portrays chaining of three classes to convert raw data into compressed and encrypted data, which is stored in a local file. The data is written to GZIPOutputStream, which compresses the input data and sends it to CryptOutputStream. CryptOutputStream encrypts the data prior to forwarding it to FileOutputStream, which writes it out to a file. The result is a file that contains encrypted and compressed data.

The source for the stream chaining shown in Figure 1 would look something like the code seen here:

```
FileOutputStream  fos = new FileOutput-
Stream("myfile.out");
CryptOutputStream cos = new CryptOutput-
Stream(fos);
GZIPOutputStream  gos = new GZIPOutput-
Stream(bos);
```

or simply:

```
GZIPOutputStream gos = new
GZIPOutputStream(new CryptOutputStream(new
FileOutputStream("myfile.out")));
```

To write to chained streams, simply call the write() method on the outermost class as shown here:

```
gos.write('a');
```

Similarily, when closing chained streams, you need only to close the outermost stream class since the close() call is automatically trickled through all the chained classes; in our example above we would simply call the close() method on the GZIPOutputStream class.

Summary

In this article we reviewed the basic concepts of Java's I/O streams, which should give you a good understanding of how to program with them. Be sure to tune in next month when we'll complete this article by looking at lots of source code to get a feel for the various uses of I/O streams such as files, databases, sockets, archives and much more.

Table 1: Data Sink Streams

CharArrayReader and CharArrayWriter	For reading from or writing to character buffers in memory
FileReader and FileWriter	For reading from or writing to files
PipedReader and PipedWriter	Used to forward the output of one thread as the input to another thread
StringReader and StringWriter	For reading from or writing to strings in memory

Table 2: Processing Streams

BufferedReader and BufferedWriter	For buffered reading/writing to reduce disk/network access for more efficiency
InputStreamReader and OutputStreamWriter	Provide a bridge between byte and character streams.
SequenceInputStream	Concatenates multiple input streams.
ObjectInputStream and ObjectOutputStream	Use for object serialization.
DataInputStream and DataOutputStream	For reading/writing Java native data types.
LineNumberReader	For reading while keeping track of the line number.
PushbackReader	Allows you to "peek" ahead in a stream by one character.

Table 3: Miscellaneous Streams (java.util.zip package)

CheckedInputStream and CheckedOutputStream	For reading/writing and maintaining a checksum for verifying the integrity of the data.
GZIPInputStream and GZIPOutputStream	For reading/writing data using GZIP compression/decompression scheme.
ZipInputStream and ZipOutputStream	For reading/writing ZIP archive files.
Stream Chaining	One of the best features of the I/O stream classes is that they are designed to work together via stream chaining.

CODE LISTING

The complete code listing for this article can be located at **www.JavaDevelopersJournal.com**

About the Author

Anil Hemrajani is a senior consultant at Divya Incorporated, a firm specializing in Java/Internet solutions. He provides consulting services to Fortune 500 companies and is a frequent writer and speaker. He can be reached at anil@divya.com.

anil@divya.com

Implementing a Grid Layout Manager with Positionable Components

Creating New Layout Managers for Specific Tasks

by **Daniel Dee**

Why Write Custom Layout Managers?

Layout manager objectifies the component layout strategy. It disentangles component layout code from the drawing code of the container of components. By isolating the implementation of a layout strategy in a separate class, a programmer can reuse it by simply assigning an instance of this layout class to the container that requires it. Without a layout manager the layout code would have to be embedded within drawing code of the container itself.

Java's AWT supports this concept in the LayoutManager interface and supplies five implementations that are sufficient in most situations. It's sometimes more efficient and effective, however, to implement one's own layout manager, designed specifically for a particular task, rather than struggling with the standard layout managers.

Let's say we need to draw components on a grid, for example, but need to specify the positions of the components arbitrarily at any time. On the first take we'd probably choose AWT's GridLayout as the layout manager – that is, until we discover that it will only position components in sequence. The programmer then would have no control over their positions. The alternative would be to use GridBagLayout, but that's way too complicated and difficult to control for such a simple task. The solution is to write our own LayoutManager. As you'll see, this isn't as difficult as it sounds and may be the best solution in many situations. This article introduces a grid layout manager with positionable components, and with flexibility and complexity somewhere between AWT's GridLayout and GridBagLayout.

Layout Manager Basics: LayoutManager Interface

AWT's LayoutManager interface declares the following methods that have to be implemented in a valid layout manager:

- addLayoutComponent
- removeLayoutComponent
- layoutContainer
- minimumLayoutSize
- preferredLayoutSize

AddLayoutComponent and removeLayoutComponent let the Container assigned with this layout add and remove a Component to and from the layout. LayoutContainer implements the layout strategy, changing the *x* and *y* pixel position and the dimension of the Components as necessary. MinimumLayoutSize and preferredLayoutSize let the Container find out the minimum required and preferred layout size given the Components to be laid out in it.

AWT's Containers are assigned default LayoutManagers but a different one may be chosen using the Container's setLayout method. When you add a Component to a Container, the Container will in turn add it to its assigned LayoutManager. The same happens when you remove a Component. Whenever a Component is added or removed, the Container will call the layoutContainer of the assigned LayoutManager to update the position and size of all Components. If you need to lay out the Container again, you may invoke the Container's doLayout method to recompute the layout.

Figure 1: Puzzle program

PositionableGridLayout Class

We call our grid layout with positionable components the PositionableGridLayout class. The minimumLayoutSize for this case should be the extent of the layout embodying the outermost component. That would be our preferredLayoutSize as well. In other words, minimumLayoutSize is the *x* grid position of the easternmost Component times the width of each grid and the *y* grid position of the southernmost Component times the height of each grid. This is shown in Listing 1.

LayoutContainer calculates the pixel location of each Component given its grid position, and relocates it with a call to the Component's setBounds method if necessary.

Note that we did not implement addLayoutComponent and removeLayoutComponent because nothing special needs to be done when a Component is added to or removed from the Container.

Listing 2 shows the implementation of PositionableGridLayout for Java 1.1.

Layout Constraints: PositionableGridConstraints Class

An automatic LayoutManager – that is, one that would not allow the programmer to control the location and size of each Component – would normally be able to work alone. For a LayoutManager that allows the programmer to individually fix the location and size of the Components, however, an assisting class is required. Such is the case with AWT's GridBagLayout, which requires GridBagConstraints to work. In our example we need to be able to position any Component at an arbitrary location at any time. Following the example of GridBagLayout, we implement an assisting class, the PositionableGridConstraints, which allows us to specify the grid position of an added component.

The PositionableGridConstraints class is

fairly simple to implement. It mainly stores away the grid position of a Component.

```
/**
* Constructs a PositionableGridConstraints.
*/
public PositionableGridConstraints( int
gridx, int gridy, … )
{
this.gridx = gridx;
this.gridy = gridy;
:
}
```

We need to assign the grid constraints to the Component, of course. Again, following the example set by AWT's GridBagLayout, we implement a setConstraints method in PositionableGridLayout whose purpose is to associate the grid constraints to a Component in a hashtable using the Component as the key, as shown in Listing 3. Listing 4 shows the implementation of PositionableGridConstraints.

Using the PositionableGridLayout Class

A good example of the use of the PositionableGridLayout class would be the familiar sliding puzzle. Initially, the tiles lie in a grid. As the user shifts the tiles' position, the application has to change the coordinate of each piece. Listing 3 shows the implementation of a simple 3x3 puzzle, the core of which follows it. Note in Listing 5 the convenience of being able to position Components by grid position instead of pixel location.

"A good example of the use of the PositionableGridLayout class would be the familiar sliding puzzle."

Conclusion

We've shown the use of LayoutManager beyond the standard AWT-supplied implementation. We've shown how customized LayoutManager can be a powerful tool for implementing unique Component layout strategy for any Container.

In a future article we'll make use of the PositionableGridLayout class and the Widget interface class featured in an earlier issue of ***JDJ*** (Vol. 3, Issue 6) to implement a TreeViewer widget, a component for laying out trees in either horizontal or vertical format.

Download Source Code

The program in this article requires the callback and widget code from the Implementing Callback and Widget-izing AWT articles published in ***JDJ*** (Vol. 3, Issues 4 and 6, respectively). Full source code (including a complete version of Puzzle) can be downloaded free from www.wigitek.com. A more extensive version supporting programmer-defined Component size, autoadjustment and alignment for Java versions 1.0 and 1.1 is also available from Wigitek Corporation at the same Web site.

CODE LISTING

The complete code listing for this article can be located at **www.JavaDevelopersJournal.com**

About the Author

Daniel Dee, president of Wigitek Corporation, holds two MS degrees and has more than 10 years' working experience in the development of GUI software toolkits. Daniel has been involved with Java since its inception. He can be reached at daniel@excaliber.wigitek.com.

daniel@excaliber.wigitek.com

Java's Dynamic Future Is Happening Now

by **David Norris**

Until recently, Java reminded me of the talented kid in school who keeps getting C's and B's instead of A's. "He's bright," the teacher says, "but he's not working up to his potential." So far, most Java developers have used this brilliant language in a tactical rather than strategic way. Java has enabled faster, easier, more flexible programming to develop traditionally structured applications within a conventional systems model.

The bright kid, though, will soon be earning straight A's as developers and IT managers reach beyond Java's role as a development tool to discover its far greater value as a strategic asset.

Without a doubt, the truly exhilarating promise of Java lies in dynamically structured applications deployed within a distributed systems model. Picture a world in which you can swiftly build distributed apps by reusing components, no matter what operating environment they were created for. Imagine calling objects on remote machines without regard to conflicting ORB standards. Contemplate building incredibly cool, distributed applications that deliver advanced functionality by commanding an army of mobile autonomous agents – smart objects that zip around the enterprise network fulfilling business missions on their own. Imagine chopping maintenance chores down to size by extending objects' behavior without having to rummage through their source code – in fact, without even needing it. And think about making this process so simple that many end users can perform their own basic maintenance, making the enterprise more nimble, and freeing your IT team to focus on strategically important business application development and systems design.

Developers and IT managers are discovering the value of Java as a strategic asset

These are the hallmarks of a dynamic distributed environment, one that is built more readily, packed with more advanced capabilities and modified with stunning speed to keep pace with ever-changing technologies and business needs.

How long will it be before you can really do all this with Java? Next year? Three years? Five? Remarkably, Java users can accomplish it all today.

A powerful new class of Java facilities, already commercially available, renders the entire vision a reality. These facilities create a layered infrastructure, a platform that leverages Java's unprecedented and underutilized ability to manipulate objects dynamically at runtime.

Many Fortune 500 companies have quietly begun constructing enterprise-scale, distributed Java solutions based on this dynamic platform – ObjectSpace Voyager. Using Voyager, they expect to shorten the application-development cycle, reduce development costs, make development teams more productive and give end users more advanced capabilities with which to conduct business.

They are also reducing the time-to-change, making the enterprise more competitive by modifying system behavior as quickly as business needs change, on the fly – a crucial strategic advantage.

That's not all. Some observers see IT careening toward another Y2K-like fiasco because many Java applets and applications are being written with a mix of business logic and application logic – precisely the programming sin that brought us the Y2K crisis itself. After all, 750,000 developers know Java; that number will soon breeze past the million mark. The sheer volume of Java development, combined with today's competitive pressures and deadlines, virtually assures that business logic is being hard-coded into some Java applets and applications. As we all know, that will stretch out the time-to-change. Someone, sometime, will have to locate each bit of buried business logic, unzip the source and change the code – unless a dynamic infrastructure is in place. With Voyager, for example, a component's behavior can be readily extended at runtime, even if the source code isn't available.

Similarly, distributed systems development has long been hindered by the struggle for supremacy among CORBA, RMI and DCOM, the leading ORB standards, all of which are incompatible. This recalls the earliest days of telephone: you couldn't place a call to the folks next door if they subscribed to a competing service.

The dynamic Java paradigm renders this problem as obsolete as a hand-cranked telephone. How? By enabling objects to interoperate with all three ORB standards simultaneously at runtime. Again, source code is never touched.

There are other benefits, too numerous to mention here. But one thing is certain: with the advent of a dynamic, distributed platform, Java is finally ready to earn straight A's – today.

About the Author

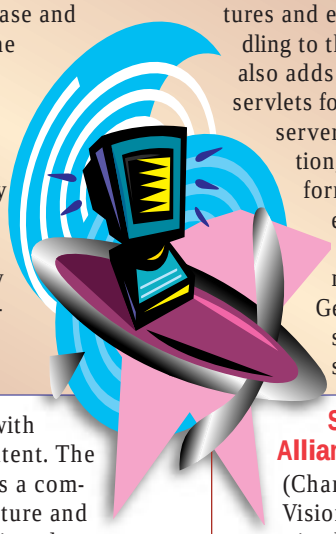
David Norris is the president, CEO and cofounder of ObjectSpace. Prior to its start in 1992, he worked extensively as an international software consultant, building distributed solutions for large corporations. David holds a bachelor's degree in computer science from the University of Texas. To contact him visit www.objectspace.com.

www.objectspace.com

New Features Added to InstantOnline

(Hermosa Beach, CA) – Gefion Software has released InstantOnline Technology Preview 1.0, a preview of a set of new components and enhancements to its InstantOnline Java Servlet suite.

InstantOnline is an alternative to proprietary server-side technologies. Supported on all major platforms, InstantOnline Basic 1.0.1 contains components for reading information from a database and presenting the result as an HTML table or as a free-form HTML pattern. They also store information submitted by Web site visitors in an HTML form.



The InstantOnline Technology Preview 1.0 adds extended variable handling, debug features and enhanced error handling to those components. It also adds a set of brand new servlets for file upload and server-side file manipulation, dynamic Web forms and for sending e-mail.

For more information visit the Gefion Software Web site at www.gefion-software.com.

Novera Releases jBusiness 4 for Enterprise Information Integration

(Burlington, MA) – Novera Software, Inc., has released jBusiness 4, the company's newest version of their information integration solution. jBusiness 4 introduces the concept of Enterprise Business Objects, a standards-based software component approach that integrates information from diverse sources while providing the scalability and manageability required of enterprise applications. With jBusiness 4, customers can use Enterprise Business Objects to create new applications that leverage data from legacy systems.

jBusiness 4 development licenses are $3,495 per developer. Deployment licenses start at $350 per concurrent user and $9,995 per server per processor.

For more information call 888-Novera1 or visit them at www.novera.com.

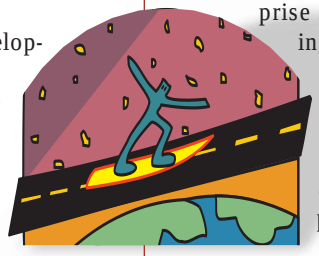
ChemSymphony Beans 1.1 Released

(Oxford, UK) – Cherwell Scientific has released ChemSymphony Beans 1.1, a suite of tools for builders of intranets with chemical content. The toolkit adopts a common architecture and has been designed to encourage object-oriented approaches to programming and data management.

ChemSymphony Beans now consists of 35 Beans that perform the most commonly needed chemical information processing tasks.

Computer Associates Unveils Unicenter TNG Framework for OS/390

(Islandia, NY) – Computer Associates International, Inc., has unveiled the Unicenter TNG Framework for OS/390, adding IBM's premier enterprise server operating system to the more than 40 hardware and software platforms that are already supported.



By delivering the Unicenter TNG Framework on OS/390, C.A. makes this platform a full and flexible peer in the enterprise. For the first time clients can position OS/390 platforms as management hubs in a heterogeneous environment or manage them from a distributed environment.

For more information visit their Web site at www.cai.com.

Sales Vision Forms Alliance with Oracle

(Charlotte, NC) – Sales Vision, Inc., has formed a strategic alliance with Oracle to deliver the industry's first 100% Java, enterprise-class sales force automation (SFA) solution using Oracle Lite as its embedded database.

Sales Vision recently ported Jsales v. 1.0 to Oracle Lite v. 3.0, Oracle's thin-client database for Java. The integrated Jsales/Oracle Lite solution provides mobile users with secure, on-demand, remote access to data residing in Oracle databases. The system runs on any computing device that supports Java and doesn't require a network connection.

For more information visit the Sales Vision Web site at www.salesvision.com.

New Course for Advanced Java Programmers

(Schaumburg, IL) – Greenbrier & Russel's training division announced the expansion of its Java curriculum, including "Advanced Java Programming," to its series of courses that provide the practical business perspective on the Java development environment.

This four-day, instructor-led course covers the following topics:

- Advanced Java Core Technologies
- Programming with Threads
- Network Programming
- JavaBeans
- JFC/Swing
- JDBC

This class is available nationwide at Greenbrier & Russel's training facilities located in Arizona, Colorado, Georgia, Illinois, Minnesota, Texas and Wisconsin.

For more information call 800 453-0357 or visit their Web site at www.gr.com.

Graph Editor Toolkit Announced

(Berkeley, CA) – Tom Sawyer Software has announced the Graph Editor Toolkit for Java. This 100% Pure Java component allows developers to build Web-enabled high-performance, scalable diagramming applications. The Graph Editor Toolkit for Java delivers proven features for building applications such as network design, workflow and process modeling. It's also fully customizable and optimized for applications running in distributed Web environments.

By using the Graph Editor Toolkit for Java, developers can efficiently produce applications that graphically reveal hidden relationships and underlying structures through ordered diagrams. These exceptionally clear user interfaces enable endusers to see and address problems rapidly.

For more information call Tom Sawyer Software at 510 848-0853, e-mail info@tomsawyer.com or visit www.tomsawyer.com.

Object Management Group

www.omg.org

JDJ NEWS

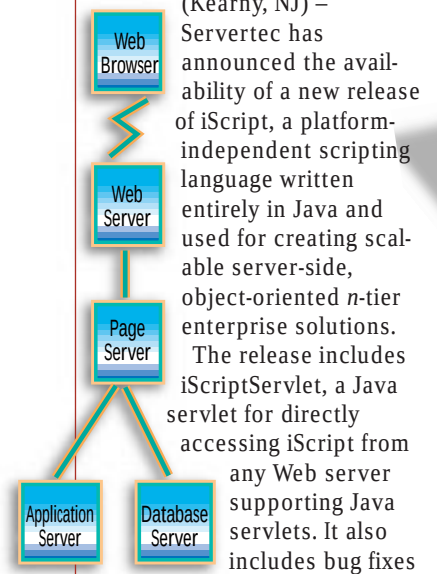
New JavaBeans Marketplace for Developers

(Cleveland, OH) - Flashline.com, an online retailer of digital products, has recently opened an online JavaBeans marketplace at www.flashline.com/Components. This software component Web store is devoted entirely to online sales of JavaBeans and other components.

Flashline's intention is to unite the fragmented component market by bringing software component buyers and sellers together and providing them with modular components designed for reuse.

For more information visit the store on the Web at www.flashline.com.

Servertec Releases New Version of iScript



(Kearny, NJ) – Servertec has announced the availability of a new release of iScript, a platform-independent scripting language written entirely in Java and used for creating scalable server-side, object-oriented *n*-tier enterprise solutions.

The release includes iScriptServlet, a Java servlet for directly accessing iScript from any Web server supporting Java servlets. It also includes bug fixes and updated documentation.

For more information visit the Servertec Web site at www.servertec.com.

Cloudscape Ships New Generation of Embeddable Database

(Oakland, CA) – Cloudscape, Inc., has shipped the first embeddable Java database designed for distributed, disconnected and mobile computing. Cloudscape version 1.5 is now available and Cloudscape version 2.0 is currently in beta with companies such as Crossroads Technologies, Endpoint! and SAIC.

The single-user development license is available for $895 and the single-user deployment license is available for $195. Five, 10 and unlimited user licenses are also available for up to $6,500.

For more information call 888 59JAVA1, e-mail info@cloudscape or visit www.cloudscape.com.

Pervasive Software Announces Pervasive.SQL for Windows CE

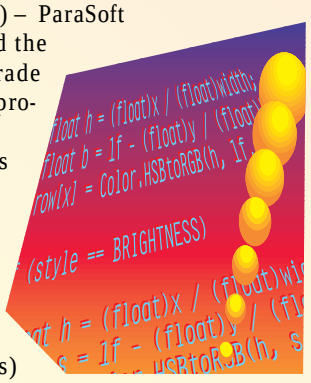
(Austin, TX) – Pervasive Software, Inc., has announced Pervasive.SQL for Windows CE– the first high-performance, multithreaded, embedded database engine for palm PCs and other smart devices.

Pervasive.SQL for Windows CE delivers Pervasive's traditional zero-administration capabilities to the Windows CE platform, offering developers the zero-maintenance features required in handheld PCs and other remotely deployed smart devices.

Pervasive.SQL for Windows CE is a full-featured, high-performance database engine built specifically for handheld devices. With a less than 50 K memory footprint, Pervasive.SQL for Windows CE is designed to deliver optimal database performance within the space constraints of the typical handheld device.

For more information call 800 287-4383, e-mail info@pervasive.com or visit www.pervasive.com.

Insure++ Gets "Ultimate Upgrade Opportunity"



(Monrovia, CA) – ParaSoft has announced the "Ultimate Upgrade Opportunity" program for Insure++. Users of competing products can upgrade to Insure++ 4.1 (on Windows NT/95/98 or UNIX platforms) and benefit from an advanced runtime error detection solution. Insure++ aims to help developers produce better quality code faster and help managers meet tight release schedules while staying within budget.

ParaSoft is offering a free, fully functional evaluation copy of Insure++ to anyone currently using a competing debugging tool. Evaluations are available from the ParaSoft Web site.

For more information call 888 305-0041 or visit www.parasoft.com.

Uniscape's Global Checker Toolsuite Available

(Redwood Shores, CA) – Uniscape has announced the addition of Global Checker for Java v. 1.0 to its Global Checker suite of high-powered code-scanning tools. Global Checker for Java 1.0 automates the multibyte enabling process and dramatically reduces the time and expense of preparing multilingual products for international markets.

Global Checker first scans C, C++ and Java-type source files for any noncompliant codes under National Language Support (NLS) standards. It then suggests solutions through an extensive online help system. This enables a developer with little NLS experience to quickly scan large C, C++ and Java files, eliminating tedious manual checking.

For more information visit www.uniscape-inc.com.

JDJ Makes Well-Received Appearance at Oracle OpenWorld '98

(San Francisco, CA) – Bonus copies of ***JDJ***'s last issue, as well as complimentary subscriptions, were extended to attendees of Oracle OpenWorld '98, held last month in San Francisco.

Java Developer's Journal was the only Java-specific publication at Oracle OpenWorld, and was the only publication officially endorsed by Oracle.

Michel Gerin Joins *JDJ* Editorial Board

(Pearl River, NY) - Michel Gerin, JBuilder product manager at Inprise, has recently become the newest member of ***JDJ***'s Editorial Advisory Board. He takes his place among an already elite selection of names within the ever-growing Java industry.

JDJ's Editorial Advisory Board members are Ted Coombs, Bill Dunlap, David Gee, Michel Gerin, Arthur van Hoff, Miko Matsumura, John Olson, Kim Polese, Sean Rhody, Rick Ross, Richard Soley and George Paolini.

Application Servers: *Part 3*

Enterprise Objects

THE GRIND

by **Java George**

In this third and final installment of our three-part quest on application server inputs, we explore the role of distributed objects. (Note the IIOP/DCOM connectivity to distributed objects in the architecture diagram below.)

There are plenty of good reasons for the application server to require access and communication with distributed objects outside its own framework. Indeed, the application server is an ensemble of distributed objects in and of itself, but there may still be external CORBA or COM objects that the developer wants to integrate into the application.

The purpose of this column is not to debate CORBA versus COM, OMG versus Microsoft or open standard versus dominant vendor. The bottom line is that both types of objects exist regardless. Now, you might ask, What will they do?

Many companies have been building and using distributed objects for important business logic, then housing those objects within Object Request Brokers and distributed object environments. At this writing it's certainly commonplace to find a mission-critical business application making use of such distributed objects. One common theme in the creation of CORBA or COM objects is the need for serious business logic in a middle tier, particularly when it provides an abstraction layer for data stored in legacy systems or legacy databases. Such distributed object abstraction layers are sometimes called *wrappers*, but I loathe the term because it's often applied too liberally – as if anything could be wrapped this way!

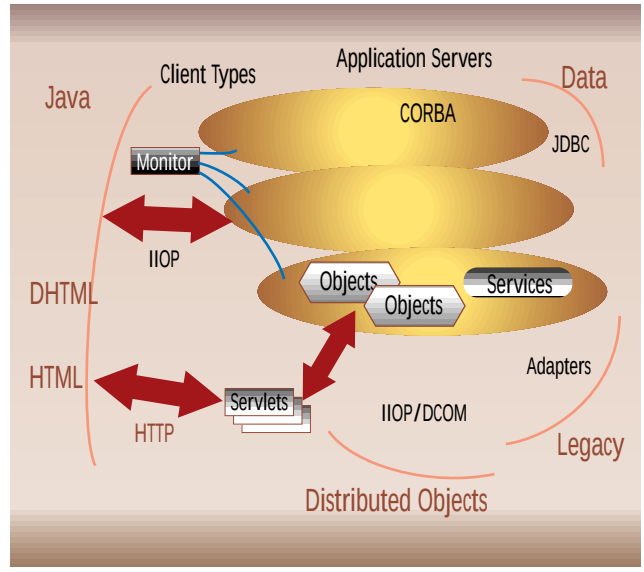


Figure 1: Architecture diagram

Given the existence of such wrapper objects and the multitude of other distinct types of distributed objects either in CORBA or COM containers, we reach the point where we must interplay these with the application server. As long as the application server itself is fluent in IIOP and DCOM, we have a place to start. Fluency in IIOP alone could suffice, however, since there are DCOM bridge products available to extend the IIOP protocol to the Microsoft realm.

> *"The application server makes a bridge so external distributed objects in separate containers can become first-class citizens of the application server framework."*

Think of the application server as a community of tightly coupled distributed objects. The application server makes a bridge so external distributed objects in separate containers (ORBs, for example) can become first-class citizens of the application server framework. That's the trick. Application server vendors such as Progress Software's Apptivity and BEA's Weblogic have efficient mechanisms for making this connection, and the result is a smooth integration between the application server and the hundreds of CORBA objects that Orfali and Harkey keep reminding us to build.

So there you have it – a brief look at the three types of connectivity for the application server: JDBC, legacy systems and distributed objects. For an application server to reach its potential, these three types must be met with elegance and purpose. It's not a matter of simply "having access" to these types of elements, for Java has access to almost everything through its APIs. We're talking about a well-thought-out, integrated approach among tools, framework and the application server.

Java George is George Kassabgi, director of developer relations for Progress Software's Apptivity Product Unit. You can e-mail him at george@apptivity.com.

George@sys-con.com