

JAVATM DEVELOPER'S JOURNAL

JavaDevelopersJournal.com

Volume: 3 Issue: 1

Looking for a
3-Tier App Builder
Tom Kim pg. 5

Blurring the Tiers
Nat Wyatt pg. 7

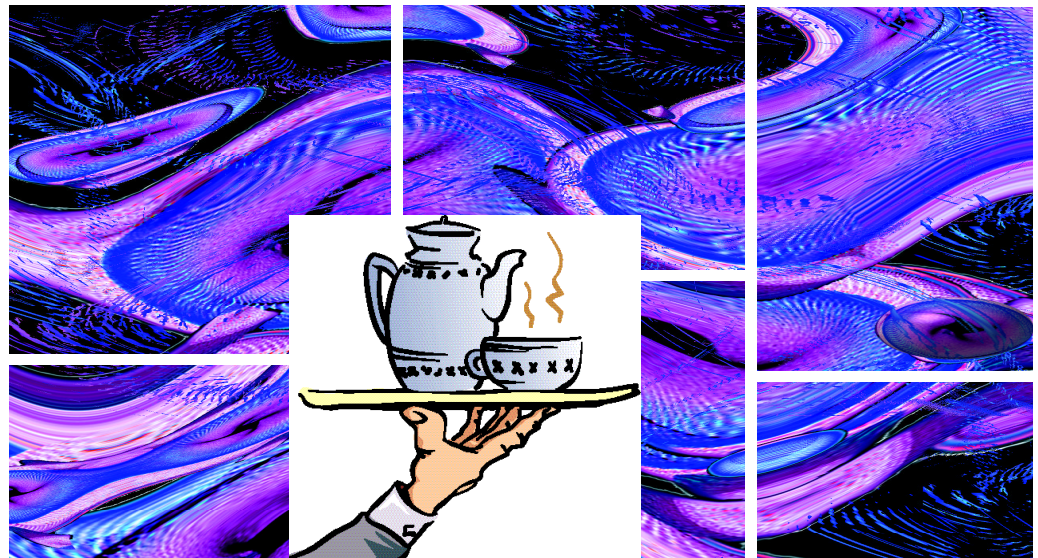
The Grind
Prophet of Doom
by Joe S. Valley pg.82

Tips & Techniques
A Simple Model/View/
Controller Pattern
by Brian Maso pg.36

Visual Café
News Today,
Gone Tomorrow
by Alan Williamson pg.64

CORBACorner
Java & CORBA
by Gerald Brose pg.60

Java News
pg.80



Migrating CGI Scripts to Java Servlets

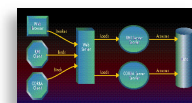
Chád Darby

Leveraging Java's OO features to build modular components

8

JDJ Feature: Servlets & Friends

Defining Java's role in the client/server hierarchy



Ajit Sagar

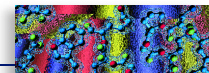
28

JDJ Feature: To Serve or Not To Serve

The impact of the servlet API on the Web community

A.R. Williamson

40



Components, Containers and Events

The platform-independent building blocks of the AWT

John Tabbone

24

URLClass Loader: The Network is the Hard Drive

Leveraging the Web to download Java Classes

Israel Hilerio

46



Developing Collaborative Games Using Active Objects

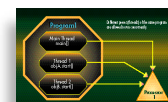
A framework for developing client and server components

Larry Chen

Todd Busby 52

Multi-Threading in Java

Using threads to solve everyday problems



Tod Cunningham

72

Stingray

Full Page Ad

Symantic Full Page Ad

Suntest Full Page Ad

EDITORIAL ADVISORY BOARD

Ted Coombs, Bill Dunlap, Allan Hess,
Arthur van Hoff, Brian Maso, Miko Matsumura,
Kim Polese, Richard Soley, David Spenhoff

Art Director: Jim Morgan
Executive Editor: Scott Davison
Managing Editor: Gail S. Schultz
Editorial Assistant: Christy Wrightington
Copy Editor: Alix Lowenthal
Technical Editor: Bahadır 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
Java Security Editor: Jay Heiser

WRITERS IN THIS ISSUE

Gerald Brose, Todd Busby, Larry Chen, Chád Darby,
Tom Kim, Nat Wyatt, Ajit Sagar, Steven Schwell,
John Tabbbone, Joe S. Valley, Alan Williamson

SUBSCRIPTIONS

For subscriptions and requests for bulk orders,
please send your letters to Subscription Department

Subscription Hotline: 800 513-7111

Cover Price: \$4.95/issue.

Domestic: \$49/yr. (12 issues) Canada/Mexico: \$69/yr.
Overseas: Basic subscription price plus air-mail 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 Manager: Claudia Jung
Advertising Sales: Diane Baird
Advertising Assistant: Erin O'Gorman
Marketing Director: Larry Hoffer
Accounting: Jennifer Patterson
Senior Designer: Robin Groves
Web Master: Robert Diamond
Web Designer: Corey Low
Customer Service: Patricia Mandaro
Rae Miranda
Sian O'Gorman

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.

© COPYRIGHT

Copyright © 1997 by SYS-CON Publications, Inc. All rights reserved. No part of this
publication may be reproduced or transmitted in any form or by any means, electronic
or mechanical, including photocopy or any information storage and retrieval system,
without written permission. For promotional reprints, contact reprint coordinator.
SYS-CON Publications, Inc. reserves the right to revise, republish and authorize its
readers to use the articles submitted for publication.

ISSN # 1087-6944

DISTRIBUTED in USA by

International Periodical Distributors

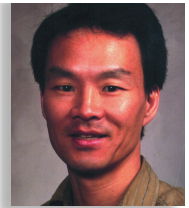
674 Via De La Valle, Suite 204, Solana Beach, CA 92075 619 481-5928

BPA Membership Applied For August, 1996

Java and Java-based marks are trademarks or registered trademarks of
Sun Microsystems, Inc. in the United States and other countries.
SYS-CON Publications, Inc. is independent of Sun Microsystems, Inc.



Tom Kim



Looking for a 3-Tier App Builder?

As we approach a new century, the Web continues its phenomenal growth rate, appealing in so many ways to diverse groups of people. With the maturation of Java, the Web will evolve from stateless HTML pages or cobbled-together HTML/CGI script solutions to true interactivity and context sensitivity. I have heard this next stage referred to as "the second Web."

To target the second Web, many people – application development tools vendors in particular – have concluded that traditional 2-tier, client/server architectures will not scale well because of the unpredictable number of user connections and data access and system administration issues.

To address the limitations of 2-tier architectures, many development groups are moving to 3-tier architectures. This is roughly defined as a presentation layer at the client tier, a server in the middle and a database of some sort on the back end. The idea is to offload the code bloat on the client or the database server, centrally manage business logic and get more flexibility out of working with the database instead of just stored procedures and triggers.

Within the last six months or so, a slew of software tools vendors have come out with 3-tier architecture development tools. However, it's becoming clear that not all 3-tier architectures are created equal.

Under the Hood

In analyzing the competition, we have looked at the various 3-tier tool offerings and have found some surprising characteristics. Many architectures are what we would call "2 1/2 tier." We call it 2 1/2 because the middle tier turns out not to be extensible at all: no APIs to add application logic exist. Essentially, the middle tier is just a database gateway. It offloads the database connectivity functions and perhaps does some database connection management producing several consequences. First, the client bears the load of initiating, querying and receiving results. Second, adding functional richness to the application adds to the size of the client.

This does not appear to be an improvement over the fat client problem, especially when we're talking about deploying the client portion of the application through browsers and over the Internet. Consider the download speeds of various client sizes over a 28.8K modem connection. In my opinion, an end-user's patience will be severely tested at 28 seconds and anything longer than a

minute will probably not be tolerated. These numbers represent just the time it takes to get the client downloaded and initialized! Once the client is running, there is also the potential for large result sets that have to be cached onto the client and then processed. All of this results in poor performance for the end-user.

And finally, adding application logic (a.k.a. business rules) to the client tier exposes your company to unnecessary security risks: Someone could easily decode your bytecode and infer information about your company and the application. If you're building applications for a competitive advantage, you risk exposing some important intellectual property.

True 3-tier

To support fast downloads of clients, protect the application logic behind the firewall and get good scalability for a Web deployable application, we need an extensible, multithreaded middle tier – the application server. Application logic would be built on top of the framework supplied by a vendor. With an extensible and scalable application server, you get the following benefits:

- Scalability: Multithreaded capability allows the server to scale well as you add more processing power, such as SMP machines and clustered systems.
- Thinner clients: By offloading processing such as database access and query results, clients can be simplified to deal only with display of information, user input and some data validation.
- Centralized application logic management: Some of this can be done with stored procedures and triggers but these have their limitations. The ability to keep business logic and business objects under a centralized group allows all clients to pick up the changes without any client administration.

We have listed only a few of the advantages to be gained by using a true 3-tier approach. When you're in the market for a 3-tier application development framework or tools, follow the wise old adage: caveat emptor. Make sure "3-tier" is really 3-tier! 🍌

About the Author

Tom Kim is the Product Manager of Java products at Rogue Wave. Prior to joining their marketing group, he worked for 12 years with IBM and Tandem Computers (6 years each) as a software engineer in the languages and tools area. Tom can be reached at

Supercede Full Page Ad

CALL FOR SUBSCRIPTIONS
1 800 513-7111International Subscriptions
& Customer Service Inquiries

914 735-1900

or by fax: 914 735-3922

E-Mail: Subscribe@SYS-CON.com
<http://www.SYS-CON.com>MAIL All Subscription Orders or
Customer Service Inquiries to**JAVA DEVELOPER'S
JOURNAL**

Java Developer's Journal

<http://www.JavaDevelopersJournal.com>**VRML DEVELOPER'S
JOURNAL**

VRML Developer's Journal

VRMLJournal.com**National JAVA
LEARNING CENTER**

National Java Learning Center, Inc.

**JAVA JOURNAL
1998 JAVA
Buyer's Guide**

JDJ Buyer's Guide

JavaBuyersGuide.com**WEB-PRO
DEVELOPER'S SUPPLEMENT**

Web-Pro Developer's Supplement

SYS-CON Publications, Inc.

39 E. Central Ave.

Pearl River, NY 10965 – USA

EDITORIAL OFFICES

Phone: 914 735-1900

Fax: 914 735-3922

ADVERTISING & SALES OFFICE

Phone: 914 735-0300

Fax: 914 735-7302

CUSTOMER SERVICE

Phone: 914 735-1900

Fax: 914 735-3922

DESIGN & PRODUCTION

Phone: 914 735-7300

Fax: 914 735-6547

DISTRIBUTED in the USA by
International Periodical Distributors

674 Via De La Valle, Suite: 204

Solana Beach, CA 92075

Phone: 619 481-5928

Worldwide Distribution by
Curtis Circulation Company

739 River Road,

New Milford NJ 07646-3048

Phone: 201 634-7400

Nat Wyatt



Blurring the Tiers

We are participating in the biggest shift in the way computers are used since the PC was popularized nearly fifteen years ago. Employees are breaking away from the confines of their offices and taking to the road. Virtual corporations are springing up and with them the demand for new types of portable computing capabilities. As integrated circuits and microprocessors get progressively smaller, computerized intelligence is popping up in unexpected places, from mini cell phones and wristwatches to smart credit cards, telephone displays and digital ink painted on walls. The functions performed by yesterday's computers and telephones are starting to insinuate themselves into our home appliances, our office furniture, even our clothing – a concept known as “ubiquitous computing.”

As mobile work forces and portable computing devices become more and more popular, we software developers are challenged to construct highly streamlined, platform-independent business applications. Fat PCs and bloated applications are giving way to slimmed-down desktop clients, lightweight computing devices and object-based software. Corporate developers are struggling to extend access to the corporate information sources that formerly could be reached only from the well-worn office chair.

This change forces companies to rethink the way software should be architected, distributed and used. Applications and databases must be small so they can be readily delivered across the Internet, and they must be portable so they can be deployed on both new and old computing devices. They have to be well connected, since data is going to have to be moved back and forth between nomadic applications and the corporate data sources.

The software fabric for these new types of computing solutions is being woven from distributed object technologies such as Java. Java programs are compact, portable and can be designed for low-bandwidth constraints. Java programs can run on any platform that supports a Java VM, including nearly every type of computer and, soon, Personal Digital Assistants (PDAs), public kiosks, smart phones and a variety of wireless devices.

What does an application look like that is designed to run disconnected, perhaps on one or more of these emerging platforms? Surprisingly perhaps, the answer is: much the same as the typical corporate applications that we deal with today. Enterprise applications these days are typically built as a two-tier or a three-

tier affair. Commentators call this n-tier architecture. The client tier looks after interaction with the user, the database tier manages data access and data integrity and between the two may be a third, application server tier that executes the business logic.

This logical three-way division of applications remains valid even when the client becomes thin or disconnected. What changes is that all three tiers may need to execute on the same client machine. At first glance, this seems strange, since some of the impetus behind multi-tiered architectures is to put the right processing on the right machine for each particular task. However, in many kinds of applications there is only one machine involved at runtime so there is no option to distribute logic to alternative locations. With only a single machine to play with, the three tiers become a logical abstraction helpful in designing the application, rather than a description of how it is implemented.

Imagine an application, written in Java, containing an embedded programmable data manager and running on a disconnected thin client. Once a day or so, the user plugs the machine into a phone line or LAN and resynchronizes with central databases. The application manages user interaction, business logic, data integrity and data access. But the boundaries between the tiers become very porous as a Java method takes a field from the user and passes it as a parameter to an order entry method that executes transaction methods against the data manager. This in turn acts as a client to an operational database at the other side of the country.

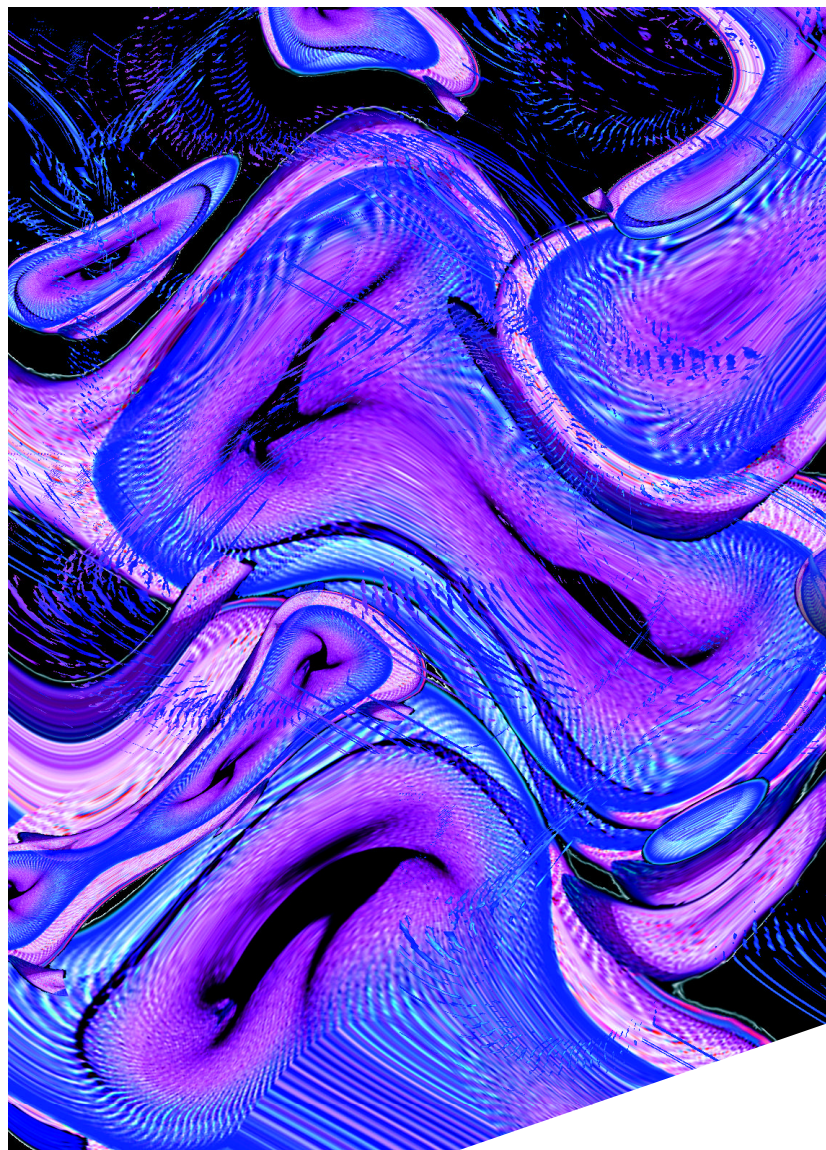
As database systems get smaller and more mobile, data management tasks can be distributed to any object, anywhere in the system, even to devices that we are not accustomed to thinking of as computing platforms. Any client can run an application server and a database manager. The fabled n-tier architecture, long sought after as the Holy Grail of distributed computing, becomes an any-tier architecture. ●

About the Author

Nat Wyatt is co-founder of Cloudscape and serves as its chief technology officer. Before founding Cloudscape, he was a senior architect and development manager at Sybase. Most recently, he led the team developing the Brahms project transactional object store. He also was a key architect for SQL Server and redesigned SQL Server to run in multi-processing environments. He can be reached at nat@cloudscape.com

Migrating CGI Scripts To Java Servlets

by Chád Darby



Introduction

The rapid acceptance of Java for client and server applications has created an immediate need to move existing CGI scripts to Java servlets. Java servlets are server-side components that can extend the functionality of a Java-enabled Web server. Currently, there are a number of enterprise Web applications that provide server-side functionality with CGI scripts. Java servlets provide a number of enhancements to the current CGI architecture.

In this article, you will learn what servlets are, how you can integrate them into your Web application and what the benefits of using servlets are. An overview of the Java servlet API is given along with servlet examples for accessing CGI environment variables and processing HTML form data.

Java Servlet Primer

What Is A Servlet?

A Java servlet is a server-side component that is platform and protocol independent. Servlets can be used to extend the functionality of a Java-enabled Web server. You can think of a servlet as a faceless applet. Servlets are loaded and invoked by the Web server in much the same way that applets are loaded and invoked by Web browsers.

When Would I Use A Servlet?

You can use a servlet to perform typical server-side processing. The servlet can communicate with the client computer and it can also communicate with other remote, networked computers. In an n-tiered environment, your middleware can be implemented as a servlet. A three-tier architecture is illustrated in Figure 1.

The following examples show how

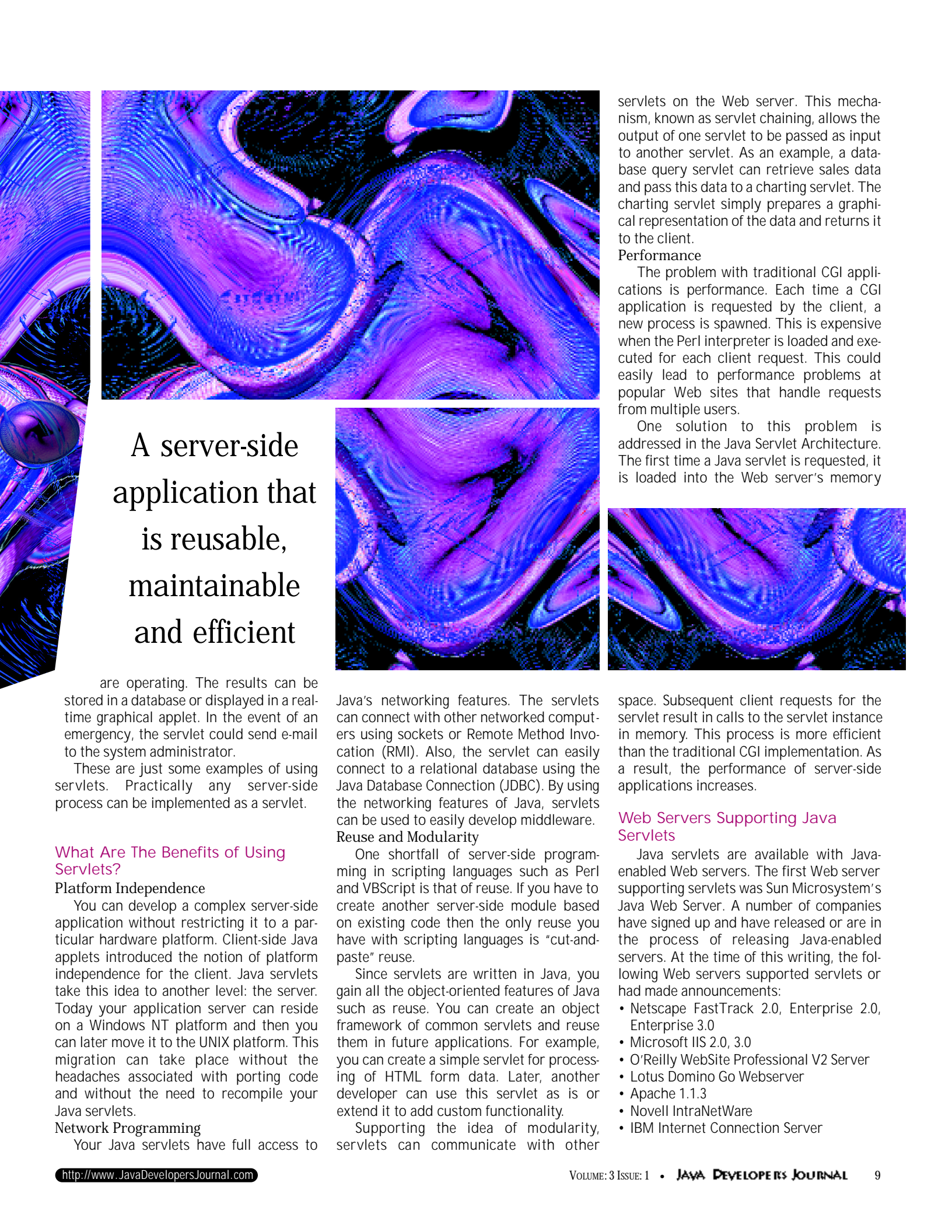
servlets can be used for data retrieval and system monitoring.

Apartment Locator Applet

The Apartment Locator applet can be used to search for apartments that match a user's criteria. The user will be able to request the city, number of bedrooms, price range, etc. The applet can send this information to the servlet for processing. The servlet, functioning as the middleware, will make the database query. The servlet can then return the query results to the applet.

Network Healthcare Monitor

An intelligent agent can be implemented as a servlet to monitor the health of your computer network. The servlets can poll host machines on your network at given intervals and ensure that the basic services



A server-side application that is reusable, maintainable and efficient

are operating. The results can be stored in a database or displayed in a real-time graphical applet. In the event of an emergency, the servlet could send e-mail to the system administrator.

These are just some examples of using servlets. Practically any server-side process can be implemented as a servlet.

What Are The Benefits of Using Servlets?

Platform Independence

You can develop a complex server-side application without restricting it to a particular hardware platform. Client-side Java applets introduced the notion of platform independence for the client. Java servlets take this idea to another level: the server. Today your application server can reside on a Windows NT platform and then you can later move it to the UNIX platform. This migration can take place without the headaches associated with porting code and without the need to recompile your Java servlets.

Network Programming

Your Java servlets have full access to

Java's networking features. The servlets can connect with other networked computers using sockets or Remote Method Invocation (RMI). Also, the servlet can easily connect to a relational database using the Java Database Connection (JDBC). By using the networking features of Java, servlets can be used to easily develop middleware. Reuse and Modularity

One shortfall of server-side programming in scripting languages such as Perl and VBScript is that of reuse. If you have to create another server-side module based on existing code then the only reuse you have with scripting languages is "cut-and-paste" reuse.

Since servlets are written in Java, you gain all the object-oriented features of Java such as reuse. You can create an object framework of common servlets and reuse them in future applications. For example, you can create a simple servlet for processing of HTML form data. Later, another developer can use this servlet as is or extend it to add custom functionality.

Supporting the idea of modularity, servlets can communicate with other

servlets on the Web server. This mechanism, known as servlet chaining, allows the output of one servlet to be passed as input to another servlet. As an example, a database query servlet can retrieve sales data and pass this data to a charting servlet. The charting servlet simply prepares a graphical representation of the data and returns it to the client.

Performance

The problem with traditional CGI applications is performance. Each time a CGI application is requested by the client, a new process is spawned. This is expensive when the Perl interpreter is loaded and executed for each client request. This could easily lead to performance problems at popular Web sites that handle requests from multiple users.

One solution to this problem is addressed in the Java Servlet Architecture. The first time a Java servlet is requested, it is loaded into the Web server's memory

space. Subsequent client requests for the servlet result in calls to the servlet instance in memory. This process is more efficient than the traditional CGI implementation. As a result, the performance of server-side applications increases.

Web Servers Supporting Java Servlets

Java servlets are available with Java-enabled Web servers. The first Web server supporting servlets was Sun Microsystems's Java Web Server. A number of companies have signed up and have released or are in the process of releasing Java-enabled servers. At the time of this writing, the following Web servers supported servlets or had made announcements:

- Netscape FastTrack 2.0, Enterprise 2.0, Enterprise 3.0
- Microsoft IIS 2.0, 3.0
- O'Reilly WebSite Professional V2 Server
- Lotus Domino Go Webserver
- Apache 1.1.3
- Novell IntraNetWare
- IBM Internet Connection Server

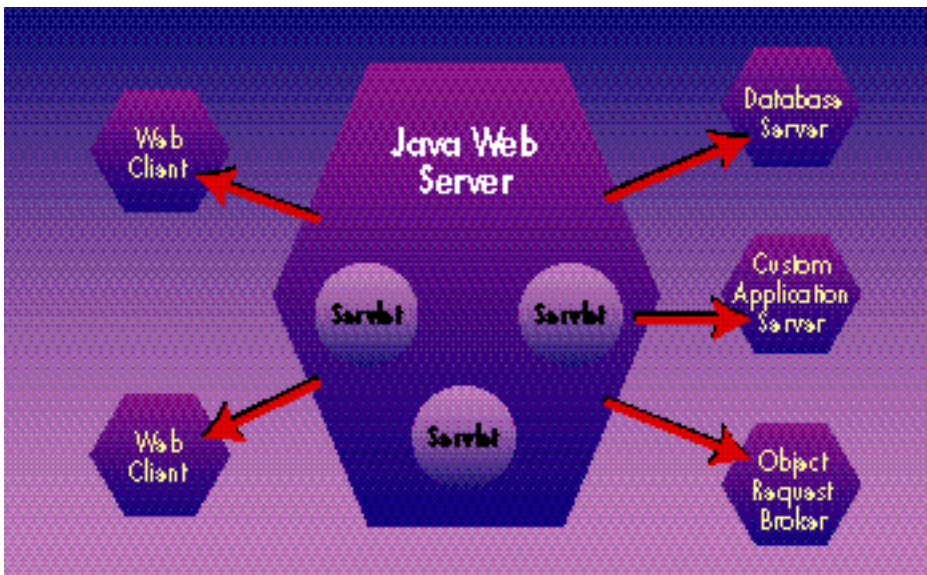


Figure 1: Three-tier servlet model

If your Web server does not support servlets then there are third-party server add-ons. Live Software offers a product called JRun. JRun is a set of classes and native code that can extend your Web server to support servlets. This product can be used to add Java servlet functionality to Microsoft's IIS and Netscape's Web servers. Visit Live Software's Web site for more details: www.livesoftware.com. It is also a good idea to check with your Web server vendor for servlet support and release dates.

Sun's Java Web Server At A Glance

You can start developing Java servlets today by downloading the Java Web Server from Sun Microsystems. The current platforms supported are Sun Solaris 2.x and Windows 95/NT. Also, the Java Development Kit (JDK) 1.1.2 or greater is required.

At the time of this writing, the final release of Java Web Server version 1.0.3 was available at Sun Microsystem's Web site: <http://jserv.javasoft.com>

The download contains simple instructions for installing and running the Web Server. You can confirm that it is properly installed and running by loading the default Web page at <http://localhost:8080/>. By default, the Web Server is listening on port 8080 as opposed to port 80. The Web server's home page has a number of useful links to developer and administrator documentation.

The Java Web Server has an administration tool that is written as a Java applet. You can administer the Web server by opening <http://localhost:9090>. The default user name is admin and the default password is admin. Once logged in, you will see the Administrator applet as shown in Figure 2.

The Administrator applet can be used to perform a number of operations such as add servlets, set up security parameters and set up servlet aliases. There are numerous other operations available and they are fully documented in the Administrator documentation. You will use the Administrator applet frequently during servlet development.

Servlet API Overview

The Java Servlet API is composed of two main packages: `javax.servlet` and `javax.servlet.http`.

The `javax.servlet` package has a collection of classes and interfaces for writing servlets that are protocol independent. The servlet can communicate with the client using a custom protocol. Tables 1 and 2 show the most useful classes. A complete

list of classes and interfaces can be found in the online documentation available with the Java Web Server.

You can use the `javax.servlet.http` package to communicate with HTTP specific client requests.

Basic Servlet Code Example

The time has come to view actual Java code for a servlet. Basically, all you have to do is extend from a servlet class. You have a choice here of extending from the abstract classes' `GenericServlet` or `HttpServlet`. You can use the `GenericServlet` class for basic server-side processing. As a Web developer, migrating CGI scripts to Java, you will most likely use the `HttpServlet` class. The `HttpServlet` class provides protocol support for HTTP 1.0. Listing 1 is a code skeleton for a basic `HttpServlet`.

First, the packages, `javax.servlet` and `javax.servlet.http`, are imported. Next, the class `TestWebServlet` is declared as public and extends from `HttpServlet`. The main entry point for your servlet is through the `service()` method. You will do the majority of your processing in this method. When your servlet is called by a client program, the `service()` method is invoked. The two parameters, `HttpServletRequest` and `HttpServletResponse`, contain client request and response information. These parameters will be used for communicating with the client.

The first parameter, an `HttpServletRequest` object, contains information specific to this client's request. In traditional CGI programming, this information could be read from the standard input stream (stdin). However, using the Java Servlet API you will access this information using an

Class / Interface	Description
<code>javax.servlet.GenericServlet</code>	Basic class for a servlet. Normally subclassed to provide a custom service.
<code>javax.servlet.ServletRequest</code>	Provides information about the client's request. You can access the client data by reading the client's input stream.
<code>javax.servlet.ServletResponse</code>	Used to respond to the client. Can send the data using the client's output stream.

Table 1

Class / Interface	Description
<code>javax.servlet.http.HttpServlet</code>	Extends the <code>GenericServlet</code> class. Adds support for the HTTP protocol.
<code>javax.servlet.http.HttpServletRequest</code>	You can access CGI environment variables and form data
<code>javax.servlet.http.HttpServletResponse</code>	Used to respond to the client. Can send the data using the clients output stream.

Table 2

ZeroG Full Page Ad

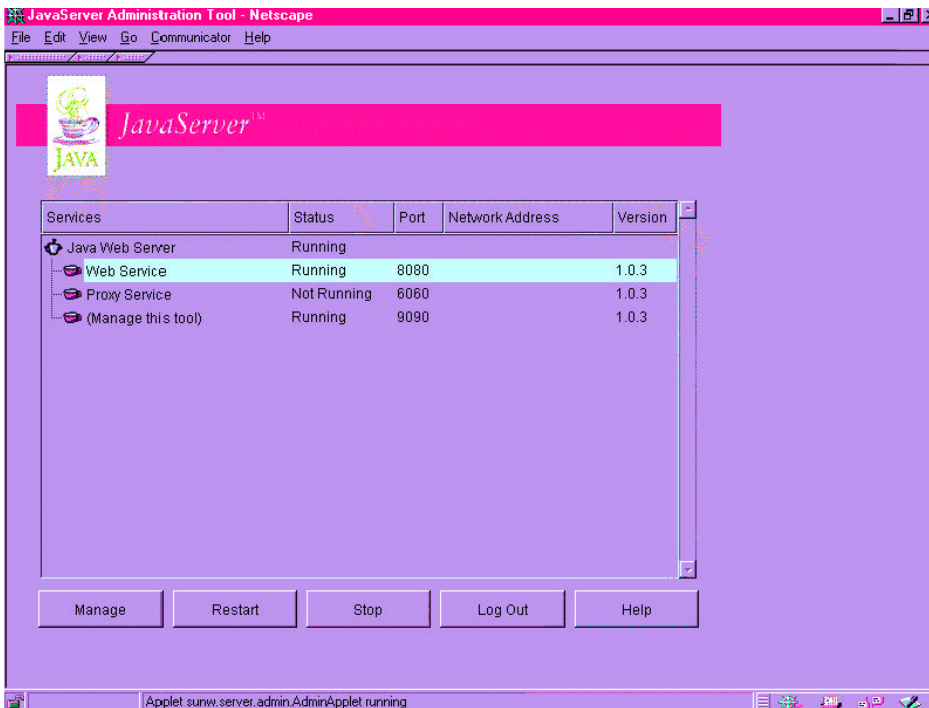


Figure 2: Main administrator applet

HttpServletRequest object. You can query this object to find out the client's host machine name and IP address. You also have the ability to find out the HTTP method used by the client to request information (i.e., GET, HEAD, POST, PUT). The most valuable method available in this HttpServletRequest object is the `getParameter()` method. This method allows you to provide the parameter name, the name of a form field for instance, and then the parameter value is returned. Here is an example of calling the `getParameter` method:

```
String userName =
request.getParameter("Name");
```

The second parameter, an HttpServletResponse object, is your vehicle for passing data back to the client. Basically, this works similarly to traditional CGI programming. However, instead of writing your information to the standard output stream (stdout) you will use the output stream provided by the client. To get the client output stream, simply call the `getOutputStream()` method on the HttpServletResponse object. With this output stream, you can construct a `PrintStream` for sending back your response. Here is an example of constructing a `PrintStream`:

```
PrintWriter out = new
PrintWriter(response.getOutputStream());
```

Now you have seen the basic framework for an HttpServlet. You are probably very eager to write, compile and run your first servlet. Well, you are in luck because now

we will discuss the steps of servlet development.

Servlet Development Process

The basic steps for developing a servlet are outlined below:

1. Write the code.
2. Compile the code.
3. Add servlet to Web Server.
4. Test the servlet.

So, let's go through this development process with a test servlet which will send a reply page to the client. The page is a simple thank you note with the client IP address listed:

Thanks for calling the servlet from 127.0.0.1

Step 1: Write the Code

Listing 2, `TestWebServlet.java`, provides the complete code for our test servlet.

Before you compile the program, take a look at the Java code. Notice that the servlet extends from `HttpServlet`. As stated earlier, the `service()` method is the servlet entry point so the bulk of the processing is accomplished here. The client's IP address is provided by calling the `getRemoteAddr()` method on the `HttpServletRequest` object. Additional `HttpServletRequest` methods will be presented in a later section. As in traditional CGI programming, the reply page is created using strings with embedded HTML tags. Next, the content-type is set for the response header using the `HttpServletResponse` object. Finally, an output stream is created and the page is transmitted to the user. It is important to note

that the output stream from the `HttpServletResponse` object is used instead of the standard output stream.

Step 2: Compile the Code

When compiling the code for `TestWebServlet.java`, you must ensure two things:

- The Java servlet classes are in your classpath.
- The .class file generated is placed in the `server_root/servlets` subdirectory.

For the classpath, the Java servlet classes are located in `server_root/lib/classes.jar`. `server_root` is the root directory where you installed the Java Web Server. To make life easy, you should modify your `CLASSPATH` environment variable.

By compiling the source code, a `TestWebServlet.class` file is generated. This file, `TestWebServlet.class`, must be copied to the servlet subdirectory located at `server_root/servlets`. If you don't want to manually copy the file after each compile, you can use the `-d` option on the `javac` command to automatically place the .class file in the specified directory. Here is an example of the command:

```
$ javac -d server_root/servlets
TestWebServlet.java
```

Step 3: Add Servlet to Web Server

Now, in order for the Web server to recognize your new servlet, you have to add it to the Web server. This is basically a registration process for the servlet.

Assuming that you are still using Sun's Java Web Server, you can follow these steps:

- If the Java Web Server is not running, start it now. Refer to the steps provided in the Installation section presented earlier in the article.
- Log on to the administrator applet by opening the following URL:

```
http://localhost:9090
user id = admin, password = admin
(default id and password)
If successful, then the main administrator
applet is loaded.
```

- Once the main administrator applet is loaded, highlight the "Web Service" and click the "Manage" button. If all works well, you should now see the "Web Service" window.
- Now click on the "Servlets" button and select "Add" from the tree in the left pane.
- In the center pane, you can enter the following information:

Servlet Name = TestMe

The Servlet Name field is a logical handle for the Web server. This name does not

Installshield Full pg

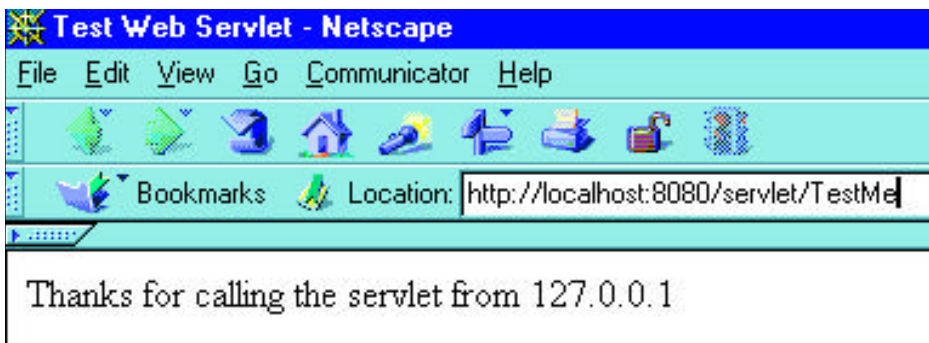


Figure 3: Output of testme servlet

depend on the name of the servlet class file.

Servlet Class = TestWebServlet

The Servlet Class field is the true name of the servlet class. Notice that you do not provide the .class extension, only the class name.

Once entered, click the "Add" button in the bottom of the center pane. Close the window.

Step 4: Test the Servlet

You can check if your servlet was properly added to the Web server by invoking the servlet. Use the following URL:

<http://localhost:8080/servlet/TestMe>

If all is well, the servlet will give you a thank you page with your IP address similar to the message shown in Figure 3.

Congratulations! You have successfully developed a Java servlet. Now you can follow the previous steps for developing your own custom servlets.

Servlet Life Cycle

By now, you are probably curious about the servlet's life cycle. A servlet has three life cycle methods: `init()`, `service()` and `destroy()`. These methods are called directly by the Web server. The life cycle methods are described in detail in Table 3.

Recall that in an earlier section we discussed the benefits of Java servlets. One of those benefits is performance, because the servlet process is spawned only once dur-

ing its lifetime. This is in contrast to traditional CGI scripts that spawn a new process each time the CGI script is invoked. As you can see from Table 3, the `init()` method is called only once to spawn the new process. Subsequent calls to the servlet results in only the `service()` method being called.

Make the Migration!

You now understand the basics of Java servlet development. In this section, you will apply this knowledge to migrate your CGI scripts to Java servlets. Example servlets will be presented for accessing environment variables and echoing form data.

Accessing Environment Variables

In your current CGI applications, you might have a need to access the CGI environment variables to process the client's request. As you will see, the environment variables provide very useful information about the client such as the browser type, IP address and host name.

The Java Servlet API provides support for accessing environment variables. You can use the methods in the `HttpServletRequest` interface to retrieve this information. The `HttpServletRequest` interface inherits the majority of its methods from `ServletRequest`.

Table 4 provides a list of CGI variables along with the corresponding methods in `HttpServletRequest`.

Method Name	Method Description
<code>init()</code>	Called only once when the servlet is invoked for the first time. You can override this method to perform typical initialization routines such as initializing a counter or making database connections.
<code>service()</code>	Called by the Web server when the servlet is requested by the client. This is the main entry point for the servlet. You will place the bulk of your servlet processing code in this method.
<code>destroy()</code>	Called when the servlet is removed from the Web server. <code>destroy()</code> is also called on each servlet when the Web server shuts down. You can use this method to clean up resource allocations and close any connections for sockets or databases.

Table 3: Life cycle methods

ECHO ENVIRONMENT VARIABLES Servlet

Let's create a simple servlet that will echo the environment variables using the `HttpServletRequest` methods.

You will use the same servlet development process presented earlier. Recall this process is: write the code, compile the servlet, add the servlet and finally test it. Let's follow this process step-by-step.

Step 1: Write the Code

Listing 3, `EchoEnvironmentServlet.java`, provides the complete source code for the servlet.

Notice that as before, a bulk of the work

CGI Environment Variable	HttpServletRequest Method
SERVER_NAME	<code>getServerName()</code>
SERVER_SOFTWARE	unavailable
GATEWAY_INTERFACE	unavailable
SERVER_PROTOCOL	<code>getProtocol()</code>
SERVER_PORT	<code>getServerPort()</code>
REQUEST_METHOD	<code>getMethod()</code>
PATH_INFO	<code>getPathInfo()</code>
PATH_TRANSLATED	<code>getPathTranslated()</code>
SCRIPT_NAME	<code>getServletPath()</code>
QUERY_STRING	<code>getQueryString()</code>
REMOTE_HOST	<code>getRemoteHost()</code>
REMOTE_ADDR	<code>getRemoteAddr()</code>
REMOTE_USER	<code>getRemoteUser()</code>
AUTH_TYPE	<code>getAuthType()</code>
REMOTE_IDENT	unavailable
CONTENT_TYPE	<code>getContentType()</code>
CONTENT_LENGTH	<code>getContentLength()</code>
HTTP_ACCEPT	<code>getHeader("Accept")</code>
HTTP_USER_AGENT	<code>getHeader("User-Agent")</code>
HTTP_REFERER	<code>getHeader("Referer")</code>

Table 4: CGI Environment Variables and `HttpServletRequest` Methods

is accomplished in the `service()` method. The basic header of the Web page is created and then the environment variables are accessed by calling the methods available in the `HttpServletRequest` class. These are the same methods listed in Table 4. Once the page is created, it is passed back to the client. As before, you can send data back to the client by writing to the client's output stream.

Also, make note of the `getServletInfo()` method. This method is available in the `HttpServlet` class and we're simply overriding it. This is a developer-friendly method that can provide information on the servlet such as the author's name, date, revision and a shameless plug for the corporate Web site.

Step 2: Compile the Servlet

SalesVision

Full pg



Figure 4: Magazine Subscription Form

Make sure you have the classpath set to `server_root/lib/classes.jar`. To compile, issue the following command:

```
$ javac -d server_root/servlets
EchoEnvironmentServlet.java
```

This will create the class file `EchoEnvironmentServlet.class` and automatically store it in the `server_root/servlets` sub-directory.

Step 3: Add the Servlet

Remember, you have to add the servlet to the Web server.

- Log in to the Web server. For details on logging in, refer to the section where you developed `TestWebServlet`.
- Double click the "Web Service" to manage it and then click the "Servlets" button.
- Now click the "Add" button and give the following information:

```
Servlet Name = EchoEnviro
Servlet Class = EchoEnvironmentServlet
```

Once entered, click the "Add" button in the bottom of the center pane. Close the window.

Step 4: Test the Servlet

You can check if your servlet was properly added to the Web server by invoking the servlet. To invoke your environment variables servlet, use the following URL:

```
http://localhost:8080/servlet/EchoEnviro
```

Your servlet should return a Web page that lists the different environment variables. Also, as a bonus, this servlet displays

the MIME-type header information.

As you can see, accessing environment variables in a Java servlet is very straightforward. So now you can easily write a Java servlet that can identify the Web browser being used by the client.

Reading The Form Data

With the knowledge you've gained so far about Java servlets, you now have the skills to process HTML form data. Forms can be used to gather user information for an order request, class registration or magazine subscription request. In this section, you will develop an HTML form and also a Java servlet to process a magazine subscription.

Step 1: Develop the HTML Form

First, we have to develop the HTML form. This form is your traditional magazine subscription form. Figure 4 is a screen shot of the form.

Listing 4, `MagazineForm.html` provides the complete HTML code for the form.

CAUTION: Form POSTing Buglet

I was migrating an enterprise CGI application to Java servlets when I ran into a bug while using the POST method to pass form data. This bug appears when using the Java Web Server 1.0.3 on the Windows 95 platform. If your form tries to POST data then you will get the following error:

```
A network error occurred while receiving
data.
Network Error: Connection reset by peer.
```

Since my project deadline was fast approaching, I used a simple work-around.

Instead of using the POST method, I used the GET method in the HTML form.

Step 2: Develop the Magazine Form Servlet

The magazine form servlet (see Figure 4) will read information from the client's submission. Once the form fields are accessed, a confirmation HTML page is returned to the user.

1. Write the Servlet Code

Listing 5, `EchoMagazineServlet.java`, provides the complete code for the Java servlet.

Notice that in this example we are overriding the life cycle method, `init()`. In this method we call the constructor of the superclass. However, the item of interest here is the initialization of the integer variable, `counter`. The counter is used to display the number of subscription forms submitted. The counter variable illustrates the fact that only one instance of the servlet is created. During repeated calls to the `service()` method, the counter is incremented.

As usual, we process the client request in the `service()` method. The counter variable is incremented in the first couple of lines in the `service()` method. Then, the basic HTML header tags are added to the `replyPage` string. A line of text is added to display the subscriber count by accessing the counter variable. Next, the form data is accessed by calling the `getParameter()` method with the name of the HTML form field. The servlet information is added at the bottom of the page by calling the `getServletInfo()` method. Finally, the servlet responds to the client by returning the constructed HTML page.

2. Compile the Servlet Code

```
$ javac -d server_root/servlets
EchoMagazineServlet.java
```

3. Add the Servlet to the Web server

Refer to the "add servlet" steps presented earlier in this article. Use the following information for Servlet Name and Servlet Class fields.

```
Servlet Name = EchoMagazineForm
Servlet Class = EchoMagazineServlet
```

4. Test the Servlet

Using your Web browser, open the HTML page, `MagazineForm.html`. Enter some sample user data and press the "Subscribe" button. A confirmation message should be returned to the Web browser.

This example uses a basic, practical approach for accessing form data. There are a number of useful methods available in the `HttpServletRequest` class. If you are building a dynamic and robust servlet for reading forms then you can call the `getPa-`

G&R

Full pg

parameterNames() routine to get a list of parameter names. The list of parameter names corresponds to the names of the HTML form input fields. The parameter list is actually returned as an Enumeration object.

The following is a simple code fragment that uses the getParameterNames() method:

```
Enumeration e = request.getParameterNames();
String name, value;

while (e.hasMoreElements())
{
    name = (String) e.nextElement();
    value = request.getParameter(name);

    replyPage += "<P>" + name + ": " +
value;
}
```

Also, if your servlet needs to read files on the Web server then you can use the HttpServletRequest.getRealPath() method. This method is useful because you can provide the relative Web file path and the absolute file path is returned. Once you

have the absolute path then you can easily create a FileInputStream or FileOutputStream.

If you want to redirect the Web browser to a new Web page then you can use the HttpServletResponse.sendRedirect() method and provide a URL string. Also, you can gracefully recover from exceptions by sending an error page. In this situation, call the HttpServletResponse.sendError() and provide the error code and error string.

As you can see, the Servlet API provides a great deal of support for communicating with the Web client.

Conclusion

Java servlets give you the capability to develop complex server-side applications. Servlets leverage Java's object-oriented features to build reusable and modular components. You can easily create servlets to replace CGI applications, access databases and communicate with remote computers.

The new beta release of the Servlet API provides support for cookies and session management. The API also features HTML page compilation with embedded Java code and HTML templates.

You can use the information in this article to migrate your existing CGI applications to Java servlets. If you want a server-side application that is reusable, maintainable and efficient then you should make the migration!

In the next article, you will learn how to build a data-aware servlet using JDBC.

References

Article Listings:

<http://www.j-nine.com/pubs/jdj>

Java Servlet API: <http://jserv.javasoft.com>

Live Software, Jrun 2.0:

<http://www.livesoftware.com>

This article is based on Java Web Server 1.0.3 final. Some details may have changed by the time you read this. ☛

About the Author

Chád (shod) Darby is a Java consultant for J9 Consulting, www.j-nine.com. He specializes in developing server-side Java applications and database applications. He also provides Java training for Learning Tree International. Chád can be reached at: darby@j-nine.com



darby@j-nine.com

Listing 1.

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class TestWebServlet extends HttpServlet
{
    public void service(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException
    {
        // get input from the client using the HttpServletRequest parameter

        // process the data

        // return output to the client using the HttpServletResponse
        parameter
    }
}
```

Listing 2.

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

// File: TestWebServlet.java
// Listing 1
//
/**
 *
 * This servlet responds with a simple thank-you note displaying
 * the client's IP address.
 *
 * @author Chad (shod) Darby, darby@j-nine.com
 * @version 1.369, 13 Nov 97
 */
```

```
 *
 */
public class TestWebServlet extends HttpServlet
{
    protected void service(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException
    {
        String clientIPAddress = request.getRemoteAddr();
        String replyPage = null;

        // Build HTML page
        //
        replyPage = "<HTML><HEAD><TITLE>Test Web
Servlet</TITLE></HEAD>";
        replyPage += "<BODY>";
        replyPage += "<P>Thanks for calling the servlet from " +
clientIPAddress;
        replyPage += "</BODY></HTML>";

        // Set the content type of response
        //
        response.setContentType("text/html");

        // Get the output stream to the client response and send
        the page.
        // Note: The PrintWriter class is a new addition to Java
        1.1
        //
        PrintWriter out = new
        PrintWriter(response.getOutputStream());
        out.println(replyPage);
        out.flush();
        out.close();
    }
}
```

Listing 3.

```
import javax.servlet.*;
```

Protoview

Full pg

```

import javax.servlet.http.*;

import java.io.*;
import java.util.*;

// File: EchoEnvironmentServlet.java
// Listing 2
//
/**
 * This servlet displays the CGI environment variables in a web
 * page.
 *
 * @author Chad (shod) Darby, darby@j-nine.com
 * @version 3.41, 2 Nov 1997
 */
public class EchoEnvironmentServlet extends HttpServlet
{
    public void service(HttpServletRequest request,
        HttpServletResponse response)
        throws IOException
    {
        String replyPage = null;

        // Build the header for HTML page
        //
        replyPage = "<HTML><HEAD><TITLE>Echo Environment
Variables</TITLE></HEAD>";

        replyPage += "<BODY>";

        // Echo the environment variables
        //
        replyPage += "<H2> Environment Variables </H2>";

        replyPage += "<p><b> Server Name: </b>" +
request.getServerName();
        replyPage += "<p><b> Server Port: </b>" +
request.getServerPort();
        replyPage += "<p><b> Server Protocol: </b>" +
request.getProtocol();
        replyPage += "<p><b> Servlet Name: </b>" +
request.getServletPath();

        replyPage += "<p><b> Method: </b>" + request.getMethod();
        replyPage += "<p><b> Path Info: </b>" +
request.getPathInfo();
        replyPage += "<p><b> Path Translated: </b>" +
request.getPathTranslated();
        replyPage += "<p><b> Query String: </b>" +
request.getQueryString();
        replyPage += "<p><b> Request URI: </b>" +
request.getRequestURI();

        replyPage += "<p><b> Remote Host: </b>" +
request.getRemoteHost();
        replyPage += "<p><b> Remote Addr: </b>" +
request.getRemoteAddr();
        replyPage += "<p><b> Remote User: </b>" +
request.getRemoteUser();
        replyPage += "<p><b> Authorization Type: </b>" +
request.getAuthType();

        replyPage += "<p><b> Content Type: </b>" +
request.getContentType();
        replyPage += "<p><b> Content Length: </b>" +
request.getContentLength();

        replyPage += "<hr>";

        // Just for kicks, let's loop thru and get the header info

```

```

using an enumeration
//
replyPage += "<H2> Request Header Information </H2>";
Enumeration e = request.getHeaderNames();
String name;

while (e.hasMoreElements())
{
    name = (String) e.nextElement();
    replyPage += "<p><b>" + name + ": </b>" +
request.getHeader(name);
}
replyPage += "<hr>";

// Build bottom of HTML page
//
replyPage += "</BODY> </HTML>";

// Set the content-type for the response header
// note: no need for extra carriage returns!
//
response.setContentType("text/html");

// finally, send this formatted HTML page back to the
client
//
PrintStream out = new PrintStream(response.getOutputStream());
out.println(replyPage);
out.close();
}

public String getServletInfo()
{
    return "Echo Form Servlet, Chad (shod) Darby, 2 Nov 1997,
www.j-nine.com";
}

```

Listing 4.

```

<!-- FILE: MagazineForm.html -->
<!-- LISTING 3 -->

<HTML>
<HEAD>
    <TITLE>Magazine Subscriptions</TITLE>
</HEAD>
<BODY>

    <CENTER>
    <H1>
Magazine Subscription Form</H1></CENTER>

    <HR><FORM method="GET"
action="http://127.0.0.1:8080/servlet/EchoMagazineForm">
    <H3>
Name and Address</H3>

    <TABLE>
    <TR>
    <TD>Name</TD>

    <TD><INPUT type="text" size=20 name="Name"></TD>
    </TR>

    <TR>
    <TD>Address</TD>

    <TD><INPUT type="text" size=20 name="Address"></TD>
    </TR>

```


Activeverse

Full pg

```

<TD>City</TD>

<TD><INPUT type=text size=20 name="City"></TD>
</TR>

<TR>
<TD>State</TD>

<TD><INPUT type=text size=20 name="State"></TD>
</TR>

<TR>
<TD>Zip Code</TD>

<TD><INPUT type=text size=20 name="Zip Code"></TD>
</TR>
</TABLE>

<H3>
Payment Method</H3>

<OL>
<LI>
Bill Me In Full <INPUT type="radio" name="Payment Method"
value="Bill Me In Full"></LI>

<LI>
Bill Me In 3 Installments <INPUT type="radio" name="Payment Method"
value="Bill Me In 3 Installments"></LI>

<LI>
Payment Enclosed <INPUT type="radio" name="Payment Method" checked
value="Payment Enclosed"></LI>
</OL>

<TABLE>
<TR>
<TD></TD>

<TD><INPUT type=submit value="Subscribe"></TD>

<TD><INPUT type=reset value="Reset"></TD>
</TR>
</TABLE>
</FORM>
<HR>
</BODY>
</HTML>

```

Listing 5.

```

import javax.servlet.*;
import javax.servlet.http.*;

import java.io.*;
import java.util.*;

// File: EchoMagazineServlet.java
// Listing 4
//
/**
 * This servlet displays the fields of a magazine subscription
 * form in a web page.
 *
 */
public class EchoMagazineServlet extends HttpServlet
{
    private int counter;

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

```

```

        counter = 0;
    }

    public void service(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException
    {
        String replyPage = null;

        // our servlet is being called, increment the counter
        //
        counter++;

        // Build HTML page's header information
        //
        replyPage = "<HTML><HEAD><TITLE>Subscription Confirmation";
        replyPage += "</TITLE></HEAD>";

        replyPage += "<BODY>";
        replyPage += "<H1>Subscription Confirmation</H1>";
        replyPage += "<HR>";

        // Thank user for subscribing and show value of "counter"
        //
        replyPage += "Thank you for subscribing.";
        replyPage += "You are subscriber <b># " + counter + "</b>";
        replyPage += "<P>We will send the magazine to the mailing
        address below.";

        // Print out the values for form variables.
        // Notice we use HTML formatting to "spiff-up" the data
        //
        replyPage += "<UL>";
        replyPage += "<LI><B>Name: </B> " +
        request.getParameter("Name");
        replyPage += "<LI><B>Address: </B> " +
        request.getParameter("Address");
        replyPage += "<LI><B>City: </B> " +
        request.getParameter("City");
        replyPage += "<LI><B>State: </B> " +
        request.getParameter("State");
        replyPage += "<LI><B>Zip Code: </B> " +
        request.getParameter("Zip Code");
        replyPage += "<LI><B>Payment Method: </B> " +
        request.getParameter("Payment Method");
        replyPage += "</UL>";

        replyPage += "<HR>";

        replyPage += "<P>" + this.getServletInfo();

        // Build bottom of HTML page
        //
        replyPage += "</BODY></HTML>";

        // Set the content-type for the response header
        //
        response.setContentType("text/html");

        // finally, send this formatted HTML page back to the client
        //
        PrintWriter out = new PrintWriter(response.getOutputStream());
        out.println(replyPage);
        out.close();
    }

    public String getServletInfo()
    {
        return "Echo Magazine Form Servlet v5.49, 2 Nov 1997";
    }
}

```

Phaos

Full pg



Components, Containers and Events

The basics of the AWT

by John Tabbone

This article focuses on the building blocks of Java's Graphical User Interface (GUI) building package, which is called `java.awt`, or just the AWT for short. AWT stands for Abstract Windowing Toolkit. It provides a platform-independent set of tools used by the Java developer to create buttons, windows, checkboxes and other common GUI elements (widgets). The base class for all of Java's widgets is called `Component`. In addition to providing a wide variety of widgets, the AWT also provides a mechanism to execute code when something happens to the widget. When the user presses your button, or selects an item from your list, an event is generated. The event contains information about what just happened to the GUI. We will see later in this article how code can be specified that will handle the events generated by a GUI.

Java provides a wide variety of GUI accoutrements, shown in Table 1.

Even though these widgets will behave the same on different platforms, they will not look the same. A GUI running on a Windows machine will have a Windows look and feel while that same GUI running on a Macintosh or Solaris environment will have a Mac or Solaris look and feel. This is because AWT widgets have **native peers**. This means, for example, that a Java Button has a corresponding native implementation. That is, the actual code that draws the button on the screen is written in C or C++ for the target machine that the Java VM is running on. Every widget has a peer that is implemented natively.

This peer architecture is completely hidden from the Java programmer. Java developers don't have to be concerned with the native implementation of widgets. Instead, developers work with the individual widget classes that make calls to their native counterparts

behind the scenes.

All of Java's widgets inherit their base functionality from the abstract class `Component`. This means that all of Java's widgets share some behaviors, states and the identity: `Component`. `Component` is a large class that defines many behaviors and it is interesting to note that there are more common behaviors defined in `Component` than differences between individual widget classes. So, even though a Button and a Choice look and behave dramat-

Checkbox	Can be either a checkbox or radio button
Choice	A drop-down choice box
List	A list of text items that can be selected by the user
Button	A button
Canvas	An area of the screen typically used to hold an image or to be drawn upon
Label	A static text label
Scrollbar	A Scroll bar
TextField	A single line area that a user can type text into.
TextArea	A multi line area that a user can type text into.

Table 1: GUI accoutrements

ically differently, there are actually more common behaviors (defined in the base class `Component`) than differences.

Look at the API documentation for class `Component`. There are many methods with confusing names. It is difficult to guess what a call to those methods will result in. Don't be alarmed. Many of the

methods defined in `Component` are meant for other classes to call, not the programmer (unless the programmer is doing some complicated operation). However, some important methods should be noted.

Since all `Components`, be they buttons or scroll bars, have a size, the behaviors that deal with size are defined in this base class. All of these methods return an instance of class `Dimension`. A `Dimension` has two data members, length and width, which are adequate to describe a size. The method `getMaximumSize()` will return the maximum size that a `Component` (widget) can be. `getMinimumSize()`, `getPreferredSize()`, will return a `Dimension` reflecting the `Component`'s desired minimum and preferred sizes. The `getSize()` method will indicate a `Component`'s current size. Method `setSize(Dimension d)` allows the programmer to size a `Component` on the screen. However, this method should rarely be used by application developers. The use of this method will be discussed in a later column on Layout-`Managers`.

It is important to realize here that since all descendants of `Component` need to manage their sizes, the behaviors are defined in this base class. This is a common practice in object-oriented programming. Leveraging **inheritance** makes classes more organized, encourages code reuse and eases debugging. Imagine if these methods were defined in every widget class. Repeating these declarations is wasteful. Furthermore, if a bug arises, the programmer only has to look in one place (the base class) to track the error.

Methods `setVisible(boolean b)`, `setBackground(Color c)`, `setEnabled(boolean b)` are other methods commonly used when programming with `Components`. They are all self explanatory and won't be explained further except to note that they too are common behaviors of all of `Components` child classes. `Component` also has several methods of the form:

```
add<SOME KIND OF EVENT>Listener( <SOME KIND OF EVENT LISTENER> listener )
```

These methods are used in event han-

PreEmptive full

ding. Java 1.1 uses a different system of event handling than the 1.0 release. In 1.1, any class can be an event listener simply by implementing an event listener interface. An event listener is an object that is interested in hearing about certain kinds of events. For example, if an object is interested in knowing whether a particular Textfield has gained or lost focus, it would implement the FocusListener interface. Components keep an internal list of event listeners that are interested in hearing about events generated by that Component. When an event is generated, the Component will call a particular method (depending on the event type) in all of the listeners kept in the internal list. Event listeners are added to the internal list by the add<SOME KIND OF EVENT>Listener(<SOME KIND OF EVENT LISTENER> listener) method. For example, if a Textfield wanted to alert a certain FocusListener when it loses focus, the addFocusListener method would be called, passing in the FocusListener as an argument. Take a look at Listing 1 for the details.

Java provides several kinds of event listeners. They are defined in the java.awt.event package. Probably the most commonly used one is ActionListener. An ActionEvent is an event that gets generated when a Component performs its primary purpose in life. For example, when a Button is pressed, an ActionEvent is generated. The ActionListener interface has only one method to implement. ActionPerformed(ActionEvent e) gets called whenever an ActionListener receives an event.

The method addActionEventListener(ActionEventListener a) is not defined in Component. Instead, a Component can add ComponentListeners, FocusListeners, KeyListeners, MouseListeners and MouseMotionListeners.

Widgets are not the only kind of descendant classes of Component. Some children of Component are used only to hold other Components. After all, what is the structure that holds Buttons, Scroll-

bars and Textareas? The other descendant of Component is called Container. A Container is a Component whose sole purpose in life is to hold other Components.

Container, like Component, is an abstract class. The children of class Container are Panel, ScrollPane, Window, Dialog, Frame and FileDialog. A Panel is an

“...it is interesting to note that there are more common behaviors defined in Component than differences between individual widget classes.”

area of the screen that can contain other Components. Panels are very common in Java and it is interesting to note that class Applet extends Panel. The reasoning behind this is that an Applet has to contain Components to display to a user. Container too defines behaviors that are common to all of its children. A Container is aware of which Components it holds, and has several getComponent methods to report this information. A Container also provides the ability to add a Component, and there are several add methods.

The add methods all take a Component as an argument and this is where things can get interesting. A Container descends Component; therefore, it can be identified as a Component. So, using the add(Component) method, one can add a Container to a Container! This is often useful when placing Components at particular locations on the screen.

Which gives rise to the next question: How does one specify a location for a Component? There are methods defined in Component that can be used to explicitly declare a location, but it was said earlier that they should not be used. Where does a Component wind up when added to a Container?

A layout manager is defined for a Container. A layout manager is a class that defines rules for Component placement. There are default layout managers for Containers so you can do some experimenting without knowing too much about layout managers, but be prepared for unpredictable results! The next column will focus on using layout managers.

Your assignment is simple. Write an applet that has only one button. Create an ActionListener and add it to the button. In the actionPerformed method, use a System.out.println(String) to prove that the button was pressed. If you want to do more, try creating a MouseMotionListener and adding it to the Panel. Use System.out.println(String) to see what happens when you get mouse events. ●

About the Author

John V. Tabbone is a lecturer at New York University's Information Technologies Institute, where he teaches two Java programming courses and advises on curriculum development. He has been a professional Java programmer since early 1996 and continues to consult on and develop systems for a variety of New York-based businesses. You may e-mail him with questions and comments at john.tabbone@nyu.edu



john.tabbone@nyu.edu

Listing 1.

```
Class MyFocusListener implements FocusListener
{
    public void focusGained( FocusEvent fe )
    {
        System.out.println( "I get printed when the Component attains
focus!" );
    }

    public void focusLost( FocusEvent fe )
    {
        System.out.println( "I get printed when the Component loses
focus!" );
    }
}
```

```
// elsewhere in the code

Textfield t f = new Textfield(),
MyFocusListener mfl = new MyFocusListener();
// Now, add the listener to the Component
tf.addFocusListener( mfl );
```

3D Graphics full



Servlets Friends

Meeting the need for performance on the server side

by Ajit Sagar

Applets were a major factor in Java's success on the client side of the Internet. While Java has had unprecedented fame on the client side of the distributed computing equation, its success on the server side is key to ensuring its survival as a distributed platform and not as just a client-side programming language.

Thought full

Java needs to address the needs of performance and robustness on the server side to fulfill its promise of providing a networking solution for the enterprise. Servlets complement applets on the server side and complete the definition of Java's role in the client/server hierarchy. In the next few years, servlets are going to play a prominent role in the acceptance of Java as the primary object-oriented language of the future.

In this discussion, I would like to focus on some of the Internet technologies and Java APIs that servlets complement or substitute for. This issue of JDJ also features other articles that focus on the Servlet API and example applications (see pages 8 and 40).

The first few sections of this article provide a high-level introduction to servlets and their role in the Java client/server architecture. Subsequent sections include brief discussions of other Java APIs and related technologies in light of the servlet paradigm.

So What Are Servlets?

Servlets are server-side Java programs that provide a means of generating dynamic Web content. They are not standalone applications that can be executed from the command line; instead, they run within Web servers. Of course, in order to run servlets inside a Web server the server must have a Java Virtual Machine running within itself. Unlike applets, servlets are not constrained by security restrictions. They have the capabilities of a full-fledged Java program and can access files for reading and writing, load classes, change system properties, etc. They are restricted only by the file system permissions, just like other Java application programs.

From a networking standpoint, servlets

are Java applications that reside on the server side of an HTTP server. Since servlets are not subject to artificial security constraints, they enable the developer to extend Java programming to the server side of the HTTP protocol.

The Java Servlet API is a standard extension to the core API.

Java Servlet Usage

Servlets may be used at different levels on a distributed framework. The following are some of the obvious examples of servlet usage:

1. Protocol support is one of the most viable uses for servlets. For example, a file service can start with NFS and move on to as many protocols as desired; the transfer between the protocols would be made transparent by servlets. Servlets could be used for tunneling over HTTP to provide chat, newsgroup or other file server functions.
2. Servlets could play a major role as part of middle tiers in enterprise networks by connecting to SQL databases via JDBC. Corporate developers could use this for several applications over the Intranet, extranet, and Internet.
3. Servlets often work in conjunction with applets to provide a high degree of interactivity and dynamic Web content generation.
4. The most common use for servlets is to accept form input and generate HTML Web pages dynamically, similar to traditional CGI programs written in other languages.
5. Servlets could be used for collaborative applications such as online conferencing.
6. A community of servlets could act as active agents which share data with each other.
7. Servlets could be used for balancing load

among servers which mirror the same content.

CGI and Servlets

Servlets provide an alternative mechanism to CGI programs for generating dynamic data. CGI programs have existed for a while; they are stable and universally accepted. They are language-independent (although they are not platform-independent). They are fairly easy to write. The question that comes to mind is, why should you consider replacing CGI with servlets? The main advantages of servlets over CGI scripts are:

Performance

Servlets offer a substantial improvement in performance over CGI. Each CGI request on the same server results in the creation of a new process. On the other hand, a servlet can continue to run in the background after servicing a request. Also, CGI programs are not threaded. Servlets can use threading to process multiple requests efficiently, provided that the JVM embedded in the Web server offers thread support.

Platform Independence

CGI programs are platform-dependent. Servlets are Java classes and follow the "write once, run everywhere" doctrine. Therefore, they are truly portable across platforms.

State Information

CGI programs are stateless because they result in the creation of a new process each time a request is serviced. A servlet has memory of its state once it is loaded by the server. The JVM running on the Web server loads the servlet when it is called. The servlet does not have to be reloaded until it changes, and a modified servlet may be dynamically reloaded without restarting the server. Maintaining state information allows multiple servlets to share information.

Applets and Servlets

Servlets are often referred to as faceless applets. Unlike applets, they have no user interface components. As mentioned earlier, servlets are free of the security restrictions that apply to applets. This is because they run within a Web server on the server-side. Thus, they are trusted programs.

Like applets, servlets may be called from HTML files dynamically and there are several cases in which the two could be used interchangeably. So when should we design servlets and when should we design applets? The answer to this question goes back to the basic issue of load distribution between the client and the server. The distributed client/server paradigm has shifted over the past few years from fat clients to

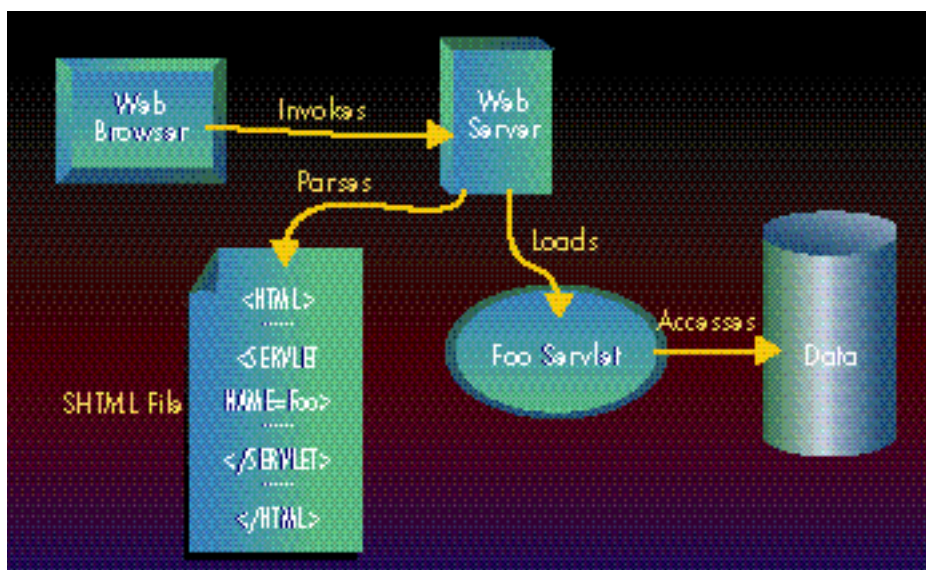


Figure 1: Servlet invocation using SHTML

Progress full

thin clients and subsequently from thin servers to fat servers. Applets are representative of the client side of the architecture and servlets represent the server side. Some scenarios in which servlets are more appropriate are given below:

1. Applet classes are downloaded over the Internet to the client and then executed in a JVM running on the client. If this involves loading large pieces of code over slow modem lines, applets are not the appropriate choice.
2. If a large part of the computation for generating the Web page can be done on the server side, it is pointless to load the part of the code that does the computation to the client. The computation should be done on the server and the results passed back to the client.
3. If processing involves operations that applets cannot perform due to security restrictions, then a local servlet may be used.

Applets are more appropriate in the following scenarios:

1. Applets basically have a well-defined user interface (remember that they derive from Panel). In servlets, on the other hand, a user interface would have to be built from scratch. Therefore, when a sophisticated user interface is desired applets are appropriate.
2. If the speed of the communication channel is adequate, then the overhead involved in downloading applets may not be an issue.

Applets and servlets can also share data and communicate. Therefore, the processing can be split between them.

Web Servers

Servlets exist only in the context of a Web server. Therefore, it is important to know what level of support servers have for servlets and what issues are involved in using one for servlet invocation. Javasoftware bundles a package with the Servlet API which may be used to embed servlet support in Apache servers, Netscape servers and Microsoft IIS. Of these, the Java Web Server from Sun is the first commercial server that incorporates the Servlet API. In Netscape servers, servlets are supported as plug-ins. In Microsoft IIS, servlets are supported via DLLs.

One thing to keep in mind while using Web servers is the version. Web server JVMs are usually a couple of steps behind Sun's current Java JDK versions. Also, the version of the Servlet Development Kit from Sun Microsystems should be supported. At the time of this writing, only the Java Web Server has complete support for JDK 1.1.

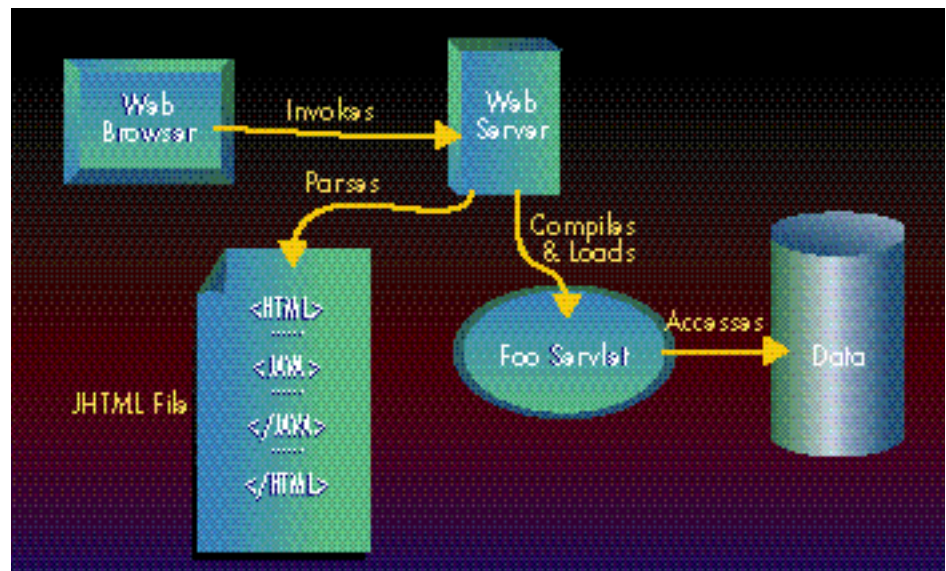


Figure 2: Servlet invocation using JHTML

Before going on to design servlets, the developers must make sure that they are working with compatible versions of the various APIs.

Server Side Includes and SHTML

Servlets can be accessed directly via a URL request from the Web client/browser. Web servers that have complete servlet support also allow servlet invocation via Server Side Includes (SSI), which involve embedding the servlet in an HTML page. The SSI mechanism is implemented in the form of an HTML tag which has the following form:

```
<SERVLET> ... </SERVLET>
```

HTML files that include SERVLET tags are called SHTML files and have the extension .shtml. The code snippet in an SHTML file shown in Listing 1 illustrates how Java servlets may be embedded in HTML.

Figure 1 illustrates servlet invocation using SHTML files.

Web Page Compilation and JHTML

As mentioned in the preceding sections, servlets are usually invoked either by a by URL invocation or as Server Side Includes in SHTML files. In both cases, the servlet is first coded as a Java class by the developer and then invoked using one of these two mechanisms. Page compilation is a different way of creating the servlet class file which involves generating HTML pages from HTML files that contain embedded Java code. These files, which are a hybrid of HTML script and Java code, are called JHTML files and have the extension .jhtml.

The Java Web Server includes support for page compilation. This is not a recommendation for readers to use the Java Web

Server but rather an introduction to an alternate mechanism for dynamic Web page design using servlets. The code snippet in a JHTML file shown in Listing 2 illustrates how Java code may be embedded in HTML.

As you can see in Listing 2, Java code within a JHTML file is embedded within <JAVA> ...</JAVA> tags. When the PageCompilerServlet is loaded, it does the following:

1. Translates the HTML/Java file into a Java source file
2. Compiles the file into a Java class file
3. Instantiates the Java class file
4. Runs the compiled file as a servlet

Figure 2 illustrates servlet invocation using JHTML files.

Servlets as Agents

Servlets may be viewed as the next step in client/server architecture towards mobile agents. Similar to an agent, a servlet may be downloaded from different servers, performing the same action on each server in turn. They also may be uploaded to different servers to perform the same function. Since they are portable across platforms, servlets can be loaded to heterogeneous platforms. Additionally, servlets bring Java's security features to the world of agents and thus help in filling the security hole that is a matter of concern in using mobile agents.

CORBA and RMI

Servlets provide the capability to access data residing on the servers in several different ways. They can be used in conjunction with distributed framework technologies like CORBA and RMI. One interesting application of servlets with CORBA is to create a servlet that is also a CORBA server. Similarly, a servlet can be made an RMI

Petronio
1/2 Ad

Softech
1/2 Ad

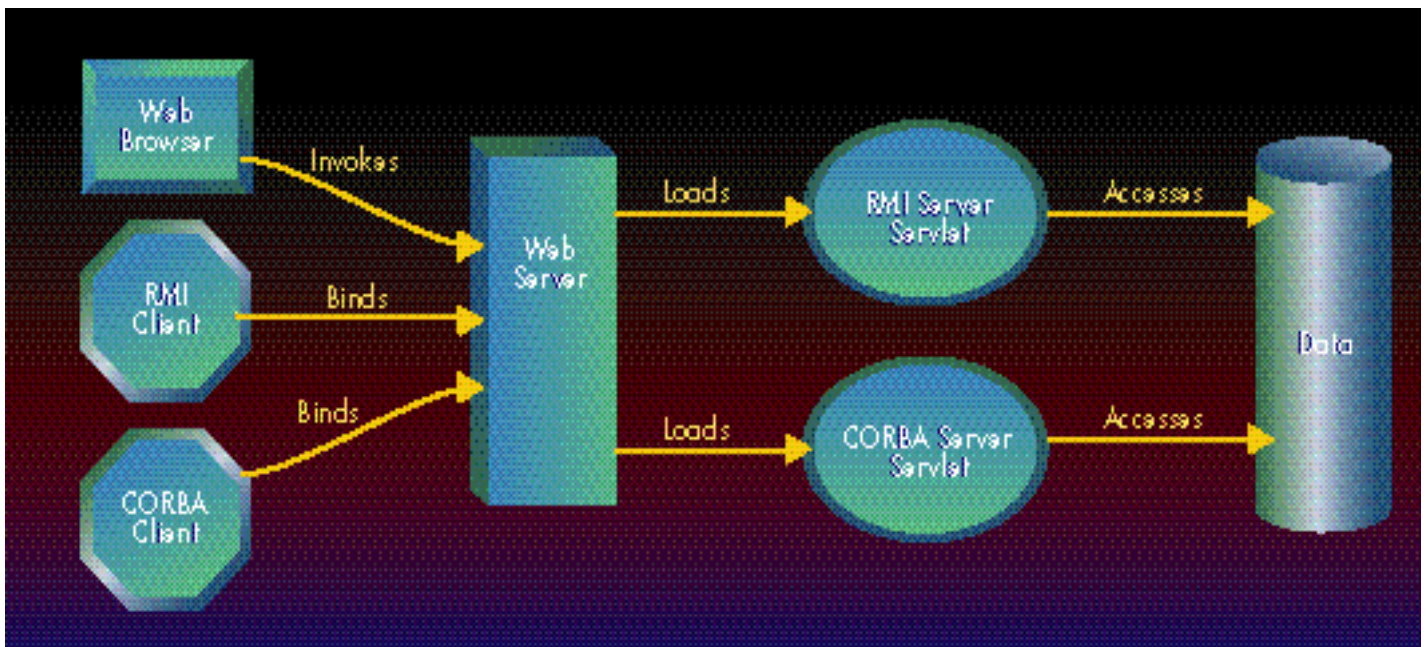


Figure 3: Web server loading two servlets – an RMI Server and a CORBA server

server. The advantage of such a servlet is that the CORBA server can accept and process Web requests directly. The client invocations on this server could be either from a Web server via HTTP requests or from a CORBA client itself.

Figure 3 illustrates a Web server that loads two servlets, an RMI Server and a CORBA server, to service client requests. Note that both servlets, once loaded, may be directly accessed by direct URL invocations via a Web browser.

Servlet Beans

Java servlet and Beans technologies can be combined to develop server-side distributed components. This involves creating servlets that are also JavaBeans™. Implementing a servlet as a Bean simply means designing a servlet that adheres to the JavaBeans design pattern. Programmatically, all this means is that the properties of the servlet that require persistence should have the Beans getter/setter methods as follows:

```
PropertyType getProperty()
void setProperty (PropertyType property)
```

The advantages of designing “servlet beans” are:

1. Any changes to the configuration of servlet beans take place immediately.
2. Beans add persistence to servlets. A servlet bean can be serialized and its dynamic state can be preserved. This way the state of a servlet can be preserved between its incarnations.

The Java Web Server from Sun Microsystems, Inc. inherently provides support for servlet beans by assuming that all servlets are beans.

Current Status

Servlets are a fairly new concept in the client/server world. They have started replacing CGI as a means for creating dynamic Web content. However, the Java Servlet API and related tools are still evol-

ing. Two key components that have been developed at Sun Microsystems, Inc. are:

1. The Java Servlet API.
2. The JavaServer Toolkit

The Java Servlet API supports Java programs on the main Web servers like Netscape servers, Microsoft IIS, etc. Also, this API is expected to be a standard extension API in JDK 1.2. The Java Server Toolkit is available in beta form and adds features to the Servlet API such as a network threading framework, dynamic Web-based remote administration, server side sandboxes, SSL, Web page compilation, client session management, etc. to servlets.

Conclusion

There are several aspects of servlets that are beyond the scope of this article – servlet security issues, servlet chaining, servlet-applet communications, accessing native code, data access, etc. Even regarding the technologies mentioned in this article, we have only touched the tip of the ice-

Listing 1: Code in an SHTML file.

```
<HTML>
.....
<BODY>
.....
<SERVLET NAME=FooServlet>
  <PARAM NAME=param1 VALUE=val ue1>
  <PARAM NAME=param2 VALUE=val ue2>
.....
</SERVLET>
.....
</BODY>
.....
```

```
</HTML>
```

Listing 2: Java code embedded in HTML.

```
<HTML>
.....
<BODY>
.....
<JAVA>
  System.out.println("Goodbye Cruel World");
.....
</JAVA>
</BODY>
.....
</HTML>
```


berg. But I hope the preceding discussion has given the readers an idea of the impact that servlets are going to have in the client/server arena and also of some of the other technologies in current distributed architectures that directly contribute to the success of Java servlets. ☕

References

Table 1 contains a glossary of the technologies mentioned in this article. Readers may also want to look up the following references for additional information:

1. The javasoft Website: <http://jserv.java-soft.com/products/java-server/documentation>.
2. Sridharan, Prashant, "Advanced Java Networking", Prentice Hall, PTR, 1997.
3. Harold, Eliot Rusty, "Java Network Programming", O'Reilly & Associates, 1997.

About the Author

Ajit Sagar is a member of the technical staff at i2 Technologies in Dallas, Texas. Ajit has seven years of programming experience in C, including one and a half years in Java. His focus is on networking, and UI architecture development. You can reach Ajit at Ajit_Sagar@i2.com

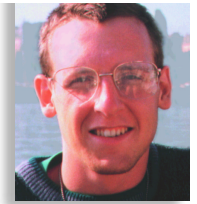


Ajit_Sagar@i2.com

NAME	DESCRIPTION
CGI	Common Gateway Interface. Defines the mechanisms by which HTTP servers communicate with gateway programs.
HTTP	An Internet client/server protocol designed for rapid and efficient delivery of hypertext materials. HTTP is a stateless protocol. It is used for transmission of all forms of data, including images, executable programs, or HTML documents.
SHTML	An HTML file with SERVLET tags. SHTML files have an .shtml extension.
JHTML	An HTML file with JAVA tags. SHTML files have a .jhtml extension.
Web Page Compilation	The process of generating HTML pages from hybrid HTML/Java files using the PageCompileServlet from the Java Servlet Toolkit
Java Web Server	A commercial Web server from Sun Microsystems, Inc. that provides complete support for servlets
Java Servlet API	A standard extension Java Extension API for developing servlets
Java Servlet Toolkit	A toolkit that adds several useful enhancements to the Java Servlet API
RMI	Remote Method Invocation. A Java-based framework for distributed Java object design
CORBA	Common Object Request Broker Architecture. An industry consensus standard that defines a higher-level facility for distributed client/server computing. Specified and maintained by the OMG
Java Beans	Java's component model developed by Sun Microsystems, Inc.
Servlets	Protocol and platform independent server side components written in Java. The Java Servlet API is a standard extension API provided by Sun Microsystems, Inc.
HTML	HyperText Markup Language. Designed specifically for marking up electronic documents for delivery over the Internet and for presentation on a variety of different possible displays

MindQ

1/2 Ad



A Simple Model/View/Controller Pattern

Using anonymous inner classes as application “glue”

by Brian Maso

In this month's column I'm going to show you how to use Java 1.1's new “inner classes” to control a windowed UI. The big advantage I've found to using this technique is that it makes creating and initializing a Java user interface simple, the technique is easy to understand and it centralizes non-reusable code (instead of spreading it across multiple classes).

I've found this technique useful in a couple of different projects, so hopefully you will find it useful, too. The first thing I'd like to convince you of is the viability and simplicity of the “Model/View/Controller” design pattern for building UIs. The idea behind this pattern is that you can decouple UI classes from reusable application classes and separate both from non-reusable application classes.

You build an MVC interface by separating the application functionality from user-interface functionality. For my example, I'm going to pretend I have an image composition application (call it IComp) that I'm trying to build. IComp allows users to load images from files and apply image filters to them.

The core of IComp, as is the case with most windowed applications, is a set of application classes that can do the “under the table” stuff that applications need to get done. For example, I might design IComp to have a generic ImageFilter class:

```
// Public interface of ImageFilter
public class ImageFilter {
    Image applyFilter(Image src);
    Property getProperties();
}
```

```
void setProperty(String name,
    String newValue);
}
```

I might also like a way to store images into a file, which I could do using an ImagePersistor class:

```
// Public interface of ImagePersistor
public class ImagePersistor {
    void init(String storageName)
        throws IOException;
    void close();
    void storeImage(String name, Image in)
        throws IOException ;
    Image retrieveImage(String name)
        throws IOException;
}
```

Both the ImageFilter and ImagePersistor classes are examples of reusable application classes. These are like vertical-market classes: classes that I would probably want to reuse in different image-processing applications.

To build the UI of the IComp application I want to reuse as many existing UI classes as possible. For example, I might want to use some of the java.awt package classes (Button, Canvas, etc.). I might also want to use some more powerful, third-party Java interface classes (a hierarchical tree control, an Image display, a color chooser, etc.).

Again, this second group of classes is going to be reusable. I might want to use them in multiple different applications I build. In the cases of both the application

classes and interface classes I want to ensure reusability as much as possible. (Remember: Reuse means improved maintainability and faster development. These are very good things.)

The “Model/View/Controller” design pattern attempts to break down application code into three distinct types of classes: those that encode a Model, those that encode a View of the Model and a Controller that manipulates both the Model and the View. In the IComp application I've been making up, the application classes are the Model. They define the state of something that is to be displayed and associated utility classes. The interface classes would be like the View. They display the Model to the user.

What's great about MVC is that it allows us to ensure the reusability of both the Model and View classes. What we've done by splitting up the application and interface classes is decoupled them from each other. I don't need to have the application classes to compile my interface classes, and I don't need my interface classes to compile my application classes. In fact, we can envision that I could use a completely different set of UI classes to build my IComp interface, but I would not have to change (or even re-compile) my application classes. That's good flexibility and proves reusability.

What's missing, however, is any code that actually makes a working application. Sure, I have interface classes I know I'm going to use and I have application classes I know my IComp application is going to use, but I don't actually have anything that “glues” them together. That's what MVC's “Controller” concept is for. The Controller is the piece that reacts to user input by manipulating the Model (like when the user presses some “Apply Filter” button in IComp's UI), or changes the View (like when the user moves one of the scrollbars attached to the view of a filtered image in IComp).

The controller classes are always going to be coupled to the Model and View

Sybex
full
new

classes. (Those purist readers who like to argue can grumble at the word “always” in the previous sentence, but it is a close enough approximation of the truth to get the idea across.) The job of the Controller classes is to “listen” to events in the UI and react to them by calling methods in the application or view objects. In order to compile my controller (“listener” and “actor”) classes, those classes need to know about the events and methods available in both the application and interface classes.

This coupling of the Controller to the Model and View classes means that usually the reusability of Controller classes is minimized. For example, I need to make a special listener in IComp that listens for the “Apply Filter” button to be hit by the user, and to actually call the applyFilter() method of a specific ImageFilter object. So, I’m not going to reuse the same listener class in another version of IComp that doesn’t have an “Apply Filter” button. So, reusability is minimal.

Java 1.1 provides us with a new tool called “anonymous inner classes” that is great to use for the role imagined by the Controller concept. Anonymous inner classes are, by their very nature, not very reusable. However, they are very easy to create. Listing 1 defines a Controller class I might make for my IComp application. The Controller object itself is responsible for listening to events that happen in the interface and reacting to those events by calling methods in any ImageFilters or UI objects that need to be manipulated. I achieve this functionality by filling the Controller class with a series of anonymous

“...you can
de-couple UI classes
from reusable
application classes
and separate both
from non-reusable
application classes.”

inner class objects, each one responsible for reacting to a particular user event.

Of course the Controller class probably is not reusable in other applications. In a very real sense the Controller class is the application. Its main() method constructs the ImageFilters (Model objects) and the UI objects (View objects) as well as a series of anonymous inner class objects to listen to and react to user events (Controller objects). What else is it an application does other than:

- Build a UI?
- Build the objects the UI is presenting an interface to?
- React to user input by manipulating the UI and/or application objects?

I generally find that I have some stable of UI classes (JFCs, third-party packages, etc.). These classes are going to be used in any of my Java applications (or applets). For a particular series of applications or applets that deal with a particular problem domain I develop a series of domain-specific classes (e.g., IComp’s ImageFilter or ImagePersistor classes might be used in any imaging application, but probably wouldn’t be used in, say, the accounting or financials arena). For each individual application or applet, I end up with very few application-specific (non-reusable) classes such as the Controller class in Listing 1.

This setup, I find, maximizes the reusability of my classes without compromising code simplicity. The MVC design pattern helps me to organize, design and debug my code in many different development environments – whether I’m using the JDK or any one of the many Java development environments out there. The design is very flexible, stable and quite scalable. ☛

About the Author

Brian Maso is a programming consultant working in California. He is the co-author of The Waite Group Press’s, “The Java API SuperBible.” Before Java, he spent five years corralled in the MS Windows branch of programming, working for such notables as the Hearst Corp., first DataBank, and Intel. Readers are encouraged to contact Brian via e-mail with any comments or questions at bmaso@develop.com.



bmaso@develop.com

Listing 1: A simple Controller class with a main() method that creates application objects and “glues” them together with several anonymous inner class objects.

```
import java.awt.*;
import java.awt.event.*;

public class Controller {
    private static ImageFilter m_filter;
    private static Image m_image;
    private static Frame m_ui;

    public static void main(
        String[] args) {
        // Create main frame window, hook up
        // listener for WINDOW_CLOSING events
        m_ui = new Frame("IComp Application");
        WindowListener wl = new WindowAdapter() {
            public void windowClosing(WindowEvent we) {
                System.exit(-1);
            }
        }
        m_ui.addWindowListener(wl);

        // Add "Apply Filter" button to UI, and
        // add listener.
        Button buttonApplyFilter =
            new Button("Applet Filter");
        ActionListener al = new ActionListener {
            public void actionPerformed(ActionEvent ae) {
                m_filter.applyFilter(m_image);
            }
        }
        m_ui.add(buttonApplyFilter);
        buttonApplyFilter.addActionListener(al);

        // ...Build interface of UI objects, and
        // anonymous inner class listeners to react to
        // user events...

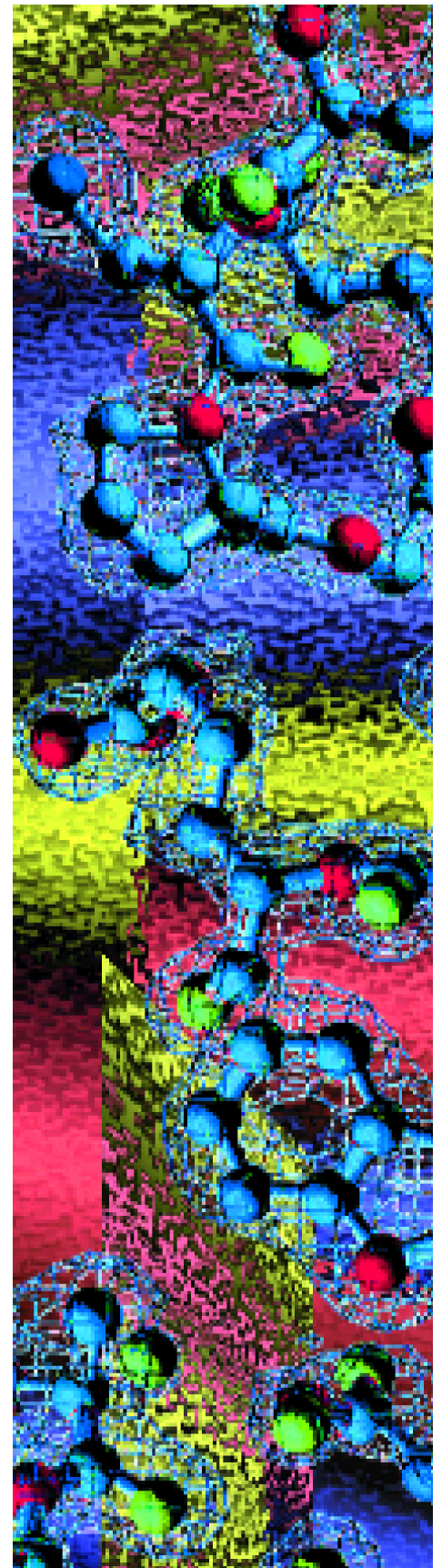
        // Finally, show the interface.
        m_ui.setVisible(true);
    }
}
```

McGraw Hill full

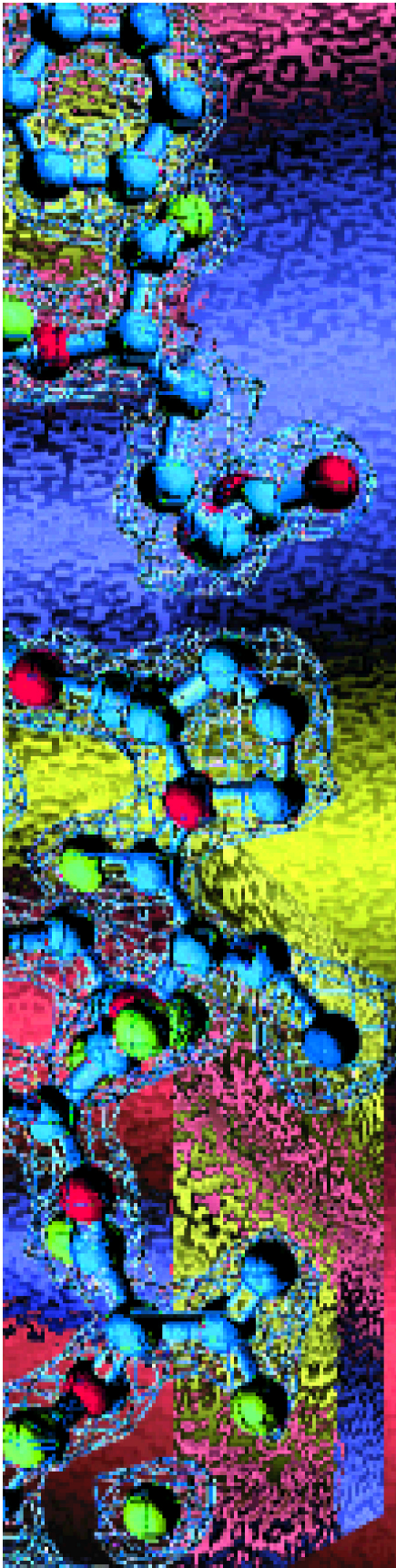
To Serve or not to Serve

That will be the Servlet

Servlets may be replacing the time
honored tradition of CGI scripts



by A. R. Williamson



Introduction

Just when you were beginning to get the hang of Java and had figured out it was more than just an animation tool, out comes yet another Java-related technology, complete with its own set of rules and conditions to dazzle and confuse. But what is so special about this new one, that we should stop and take note of it?

Up until now, Java has been very much associated with the client side of the client/server equation, popping up in applets and Beans. The server side has been relatively untouched. In fact, for a while it looked as if the Java community had forgotten all about the server, determined to take over the world with applets. But now that is about to change. The Java Servlet API has arrived and world domination is just around the corner!

OK, I may be a little bit passionate, but the Servlet API is going to make an enormous impact on the Web community and especially the client/server arena. This article addresses what the Servlet API actually is and where it can be used. Towards the conclusion, we will look at implementing an example program that demonstrates the power of this new technology.

Servlets

So what is the Servlet API? A servlet is a small program that runs in response to a client connection or request. The server facilitates the connection process with the client and, once the connection has been established, hands over the processing to a servlet.

As a demonstration of its power and flexibility, JavaSoft has developed a Java-based Web server, JWS. JWS is a commercially available Web server supporting all the features currently found in mainstream Web servers but with the addition of supporting servlet execution. Fortunately, JWS is not the only Web server to be supporting the servlet API. At the time of this writing, Apache, Netscape and Microsoft have released extensions to their existing Web servers to support servlets.

Note: The term 'servlets' is not unique to JavaSoft's implementation. Netscape has its own Servlet API that works very similarly to JavaSoft's offering. However, this version, although Java-based, is supported only on Netscape's own Web and Enterprise Servers and no plans for other server vendors to support it have been announced.

Servlets are to the server what applets are to the client. They extend the functionality offered by the server just as applets enrich the browser environment.

For this reason, this article concentrates on the servlet API, demonstrating the

power this new set of classes has over existing technologies.

A Better CGI?

The CGI (Common Gateway Interface) standard is a set of rules governing the collection and passing of data to back end programs. For this reason, CGI was well suited to the Web community. CGI gave the ability to create dynamic Web pages, depending on input from the user. For example, HTML form processing is commonly implemented with a CGI script. Another popular area in which CGI has been dutifully employed is hit counters – counting the number of visits to a particular page or resource.

So, servlets are just another alternative to the CGI interface? If only the world was as simple as this, wouldn't it be a wonderful place to live? In the most rudimentary definition, the answer would be yes. However, to appreciate the servlet API, one has to look at what CGI actually is and what its shortcomings are.

The CGI interface does not specify the underlying programming language with current script implementations found in PERL, C and C++, to name a few. Every time a CGI script is called upon to execute, a new program instance is created. This involves loading the code, allocating memory and then executing the program, taking the standard output and passing it straight back to the client. This takes a finite amount of time, and with hundreds and possibly thousands of simultaneous requests, the system can become clogged up with processes very quickly. This is because for every new connection, a new program instance is created.

Servlets, on the other hand, operate differently. When a client connection comes in, the servlet is loaded into memory and then run. However, the servlet is not removed. Subsequent requests are run with a simple method call, with multiple requests being handled by the same servlet under the Java multithreading environment. The server maintains a pool of threads and then allocates a thread of execution to each client request that comes in. If a thread is not available, the client blocks until one does become free. This approach has many advantages.

The server never becomes uncontrollably consumed with processes. A fixed number of threads is set by the administrator, which eliminates the scenario in which 1000 client requests equates to 1000 separate processes being created, using up memory and processing cycles. The need to continually reload and initialize programs is completely eliminated, using a 'load-once, run-many' policy.

Description	CGI	Servlets
Serve client requests	✓	✓
Execute third party software	✓	✓
Rely on third party software	✓	X
Client request = separate program instance	✓	X
Can it consume servers resources	✓	X
Support Server Side Includes	✓	✓
Efficient	X	✓
100 % Portable across platforms	X	✓
Standard library	X	✓
Rich in security features	X	✓

Table 1

To be fair to the CGI community, they had foreseen this shortcoming long before servlets were on the agenda. To address this, they came up with a new standard called FastCGI. This standard insisted on CGI programs being written in C and, instead of being loaded every time, the Web server would cache them in memory. Using C ensured the fastest possible execution time and by caching the program, the load overhead was eliminated. While significantly improving the CGI performance, as you will discover, FastCGI is still far short of the offerings from the servlet based solution.

Since CGI is a set of rules on how to move data from one point to another, it has no standard libraries or methods as such. For this reason, CGI scripts rely heavily on third party programs to do the majority of their work. For example, when an HTML form has been posted, the most common action to be taken is to mail the data to an e-mail account somewhere on the Internet. To achieve this, the CGI script must package the data into a file and then call upon some other program to perform the actual sending. As you can see, this is not really that efficient as not only does the server have to load and initialize the CGI script itself, it now has to load another program and start its execution. Suddenly the 1000 client connections results, in a worse case scenario, of 2000 processes running.

Servlets are implemented in Java, which comes with a rich set of standard libraries that ensure the majority of tasks are kept inside the Java environment and do not require the assistance of any other program.

Another area of hot debate is the portability of CGI. Many claim it to be very portable between different platforms. However, if a script relies on a third party piece of software, such as the standard 'sendmail' program used for sending e-mails in the UNIX environment, how can this be

portable? In order for it to work on an NT system, a 'sendmail' utility would need to be available somewhere on the system with exactly the same operational characteristics. For this reason, CGI may never be 100% portable across platforms.

Java, on the other hand, is Java; a true multiplatform language that relies on no underlying operating system. If everything is implemented in Java, then porting is no longer an issue. It is merely a question of copying the bytecodes (or .class files) to the server and then running. No recompiling and no awkward path names or environment variables to wrestle with. To many, this feature alone is enough to make servlets the winner hands down. Table 1 summarizes the differences between the two methods of processing client connections.

'Hello World'

Now that you have seen why you should be using servlets, let's take a look at how you use servlets and what the class hierarchy is.

All servlets are derived from the GenericServlet class from the javax.servlet package. This class defines all the functionality required in processing a client connection. Listing 1 shows the most famous program in the world: 'Hello World'.

As you can see, the servlet has two methods: `init(...)` and `service(...)`. The `init(...)` method is called once at servlet initialization. In this method, all global variable initialization can be performed, or if the servlet has a state change, then the loading of a configuration file can be performed here. The `init(...)` is not required to be overridden by all implementations.

The `service(...)` method is where all the work is performed. For each client connection that is accepted, this method is called. This is why a servlet implementation is much more efficient than a CGI implemen-

tation. Instead of a new program being created for each client request, a method is simply run.

A servlet has an input and an output stream. The input stream is where data from the client is passed. For example, a POST or a GET may contain information from the client as a result of a form posting. The output stream is the data that is sent back to the client and is typically displayed in their browser. Two classes have been designed to support this mechanism: `ServletRequest` and `ServletResponse`.

Those of you familiar with the Java environment will know all input and output is performed using Streams, not unlike the Streams used in C++.

```
InputStream in = _Req.getInputStream ();
OutputStream out = _Res.getOutputStream ();
```

Having a reference to both streams means any of the standard stream handlers can be used for communication. For instance, in our 'Hello World' example we used the `PrintStream` to give us a very easy means to send strings to the client. We then used it to send out a very simple HTML page with the words 'Hello World!!!' embedded in it.

This example, although not exactly rocket science, demonstrates the use of servlets in creating dynamic HTML pages. This servlet is referenced just like any other resource on the Internet – through the use of a URL. For example, this servlet can have an alias set up to it, which would result in it being accessed using:

<http://www.n-ary.com/helloworld.html>

The next example will look at a real life problem and show how the servlet is best suited for the task of providing server side solutions.

HTML Form Example

To illustrate servlets in more detail, let's take a look at a very simple servlet that will take all the fields filled in on an HTML page and e-mail them to the user `admin@some-where.com` (see Listing 2). For convenience we have employed the services of the `sun.net.smtp.SmtpClient` class from the Sun libraries, which forms part of the Java Web Server release. The use of these classes are not recommended for production software as they are liable to change in the future, but for this example they will be perfect.

The `form2email` class is based on the `javax.servlet.http.HttpServlet` class, since we are servicing an HTTP request. Since this servlet requires no initializing, the

init(...) has not been overridden but the service(...) method has. It is here that the client request comes in.

The first part of the service method deals with setting up the outgoing mail; setting the to, from and subject fields of the mail message. The next stage retrieves all the parameters that have been sent by the browser to the servlet. Each parameter can be addressed using the `HttpServletRequest.getParameter(...)` method, using the name of the variable as defined in the HTML file as the input. However, in this example prior knowledge of the variable names is unknown. Not to worry. The `HttpServletRequest.getParameterNames()` returns an Enumeration to a list of variable names that are contained within the request. By looping around and calling the `getParameter(...)` method, we can collate all known parameters from the client.

With the construction of the e-mail message and having sent it, the final task to complete is to inform the client that the request was dealt with. We will send out a simple Web page thanking them for filling out the form. As you can see in Listing 2, this is very easy. We set the MIME type of the outgoing data, get the output stream and then write the lines of text to the output stream. That's it.

Although it is a very trivial example, this serves to illustrate the ease by which servlets can be used to process client requests.

A Day in the Life of a Servlet

A servlet is loaded once and remains in memory until the server is restarted. Although on the whole this is true, a servlet can be loaded and unloaded on demand if required. This is controlled through the NAME attribute of a servlet. If a servlet has been given a logical or symbolic name, then it will remain in memory and every time a call to the same logical named servlet is made, the servlet's `service(..)` method will be called. However, if a servlet has been given no name, after execution it is removed from memory again, thus rendering the operation along the same lines as the CGI alternative.

Future of the Server

In today's heavily used Web environment, the need for more and more processing is becoming quite evident. However, at what side of the equation should this processing be performed: the client or the server?

Fortunately, answering this question is a lot easier when you take the current constraints of the Internet into consideration. At the moment, the limitation in bandwidth dictates that the amount of data being sent to the client should be kept smaller instead

or larger. For this reason, the Java Applet is not always the best solution. How many times have you sat looking at a grey rectangle, waiting for the applet to load and begin execution? This introduces an unnecessary waiting factor and tugs at the patience of the user.

The reason the growth in applets took place was to increase the functionality at the client side while reducing the load at the server end. This was before servlets came onto the scene. While a lot of the areas where applets are used cannot be

details of the connection. This is handled by a JDBC driver, supplied by the database vendor. By employing this technique the database can be switched in and out without replacing or recompiling any code elsewhere in the solution. The servlet can either send the data back to an applet to display, or alternatively create dynamic HTML pages that not only load quicker, but are faster to process.

This is but one area where a complete Java solution is now practical. On a smaller scale, servlets can be employed every-

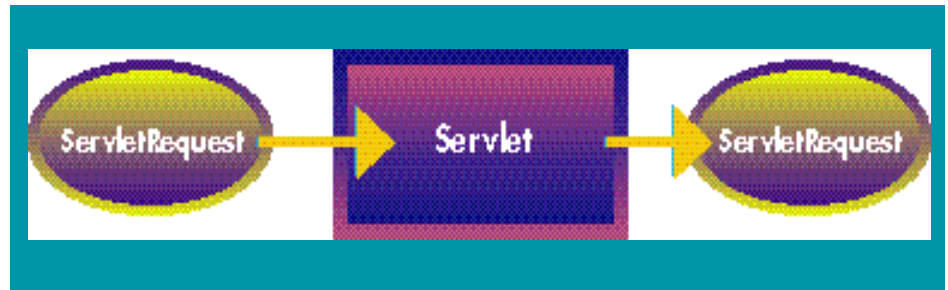


Figure 1

placed back onto the server, there are a lot of applications that could benefit from a servlet implementation instead of an applet solution.

The applet is bogged down with the speed of loading. Applets are growing in size, with the average applet size ranging from anywhere between 10k to 200k. This, in today's Internet environment, is unacceptable to the majority of users. HTML is fast because the actual data transferred is very small. This results in pages being constructed as the data starts arriving and not after all the data has arrived, as with the applet.

Areas that can benefit from a redesign, include database front-ending. Typically, solutions involved a Java applet talking to a CGI script or custom server application to retrieve data from the database. This solution is fraught with problems. Not one section of the solution could stand a change from another section. For example, if the database were to be replaced with a much bigger system, the CGI scripts would need to be redeveloped. This has a knock, in effect, to the Java applet that is front-ending the whole system.

Before servlets, providing a complete Java solution was not that practical. Something still had to take the request from the client and pass it on to the database, and this was either a custom built server or reliance on a CGI solution. Now a complete Java solution is possible.

Servlets can be employed to talk to the client and the database using the JDBC (Java Database Connectivity) interface. JDBC gives a generic interface to the database without worrying about the actual

where CGI solutions were implemented. These include:

- Search engines
- Form processing
- Page Counters
- Push technology
- Random links
- Localization of pages
- HTML filters
- HTML-based news groups
- HTML chat
- Guest books
- Live video/music feeds

Summary

This article presented an overview of the new Servlet API available from JavaSoft. Looking at the past history of Java, it won't take long before we start seeing servlets replacing the time honored tradition of the CGI scripts.

Developers can now develop custom server side applications without worrying about the internal details of each platform. They can get on with solving the problem and not 'work around' to each individual operating system. 🍌

About the Author

Alan Williamson is on the board of directors at N-ARY Limited, a UK based Java software company, specializing in Java/JDBC/Servlets. He has recently completed his second book which focuses on Java Servlets. His first book looked at using Java/JDBC/Servlets to provide a very efficient database solution. Alan can be reached at alan@n-ary.com (<http://www.n-ary.com>). He welcomes suggestions and comments.



alan@n-ary.com

Listing 1: Hello World.

```
import javax.servlet.*;
import java.io.*;

public class HelloWorldServlet extends GenericServlet
{
    public void init(ServletConfig _Config)
    {
        super( _Config );
    }

    public void service ( ServletRequest _Req, ServletResponse _Res )
        throws IOException
    {
        _Res.setContentType( "text/html" );

        PrintStreamOutput = new PrintStream( _Res.getOutputStream() );

        Output.println( "<HTML> <HEAD> <TITLE>Hello World</TITLE>
<HEAD>");
        Output.println( "<BODY>" );
        Output.println( "<H1> HELLO WORLD!!!! </H1>" );
        Output.println( "</BODY> </HTML>" );
        Output.flush();
    }
}
```

Listing 2: form2email.java.

```
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import sun.net.smtp.SmtpClient;

public class form2email extends HttpServlet
{
    public void service(HttpServletRequest _req, HttpServletResponse
_res)
        throws IOException
    {
        PrintStream    OutMail;
        SmtpClient      sendmail;

        try
        {
            ///-- Setup the outgoing mail
            String TO      = _req.getParameter("admin@somewhere.com");
            String FROM     = _req.getParameter("admin@somewhere.com");
            String SUBJECT  = _req.getParameter("Web Form");

            sendmail = new SmtpClient("mail.somewhere.com");
            sendmail.from( FROM );
            sendmail.to( TO );

            OutMail = sendmail.startMessage();

            OutMail.println("From: " + FROM );
            OutMail.println("To: " + TO );
            OutMail.println("Subject: " + SUBJECT );

            OutMail.println( "\n*****\n" );

            ///-- Retrieve all the HTML fields from the input stream
            Enumeration keys;
            String key;
            String value;
```

```
keys = _req.getParameterNames();
while (keys.hasMoreElements())
{
    key = (String) keys.nextElement();
    value = _req.getParameter(key);
    OutMail.println( key + " = " + value);
}

///-- Complete the mail, and send it
OutMail.print("\r\n");
OutMail.println( "\n*****\n" );

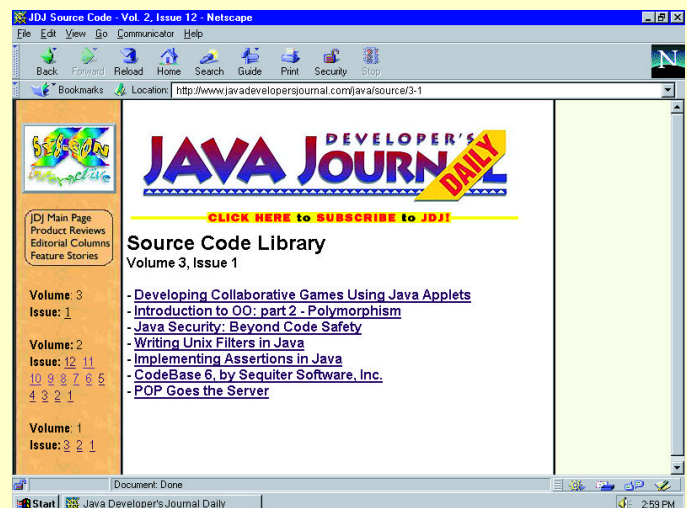
OutMail.flush();
OutMail.close();
sendmail.closeServer();
}
catch ( IOException E ){}
```

```
///-- Display thank you page to the client
_res.setContentType("text/html");
PrintWriter Out = new PrintWriter( _res.getOutputStream() );

Out.println( "<HTML><BODY>" );
Out.println( "<B><I>Thank you</I></B>" );
Out.println( "<BR><BR><B>...for taking the time to fill out
this form</B>" );
Out.println( "<BR><BR><B>A response has been sent.</B>" );
Out.println( "</BODY></HTML>" );
Out.flush();
}
}
```

Don't Type it...
Download it!

Access the source code
for this and other articles
appearing in this issue at
JavaDevelopersJournal.com



Mecklermedia full



URLClassLoader

The Network is the Hard Drive

Leveraging the Web to download classes

by Israel Hilerio

One of the primary reasons for the success of Java has been its robust dynamic class loading mechanism. The Java Virtual Machine ClassLoader is a mechanism that Java uses to load classes at runtime. Techniques to take advantage of Java's dynamic nature and the interactive nature of the Web usually create a spark of interest in the Java community.

This article leads the reader through the development of a new Java ClassLoader mechanism that leverages the Web for downloading Java classes. It begins with an overview of the Java ClassLoader mechanism, followed by an examination of how Java's ClassLoader can be extended to download Java classes from multiple Web servers to be used by a single Java application. This will allow users to develop applications that download behavior dynamically and extend their functionality on the fly using the Web.

The Mechanics of the Java ClassLoader

The Java Virtual Machine ClassLoader is a mechanism used to load classes from files in directories or zip files specified in the `java.class.path` system property. This property is set using the `CLASSPATH` environment variable. Classes are searched sequentially through the directories and zip files contained in the `java.class.path` property. The default setting for the `java.class.path` property is the subdirectory where the JDK has been installed and the zip file containing the Java classes (i.e., `classes.zip`). The ClassLoader is responsible for taking a class name and loading it into the Java runtime process.

Another important activity of the ClassLoader is to link a Class. Linking takes a binary class and incorporates it into the Java Virtual Machine Runtime. This allows a class to be executed. There are three distinct operations involved when linking:

1. Verification
2. Preparation
3. Resolution of symbolic references

Verification ensures that the binary representation of a class maps to a valid operation, the instructions are type safe and the integrity of the Java Virtual Machine is not violated. Preparation initializes the static fields (both variables and constants) of a class with default values, and checks for abstract method consistency between classes. The resolution of symbolic references is used to verify the correctness of a reference (method signature verification, field signature verification, etc.) and substitute symbols with direct reference.

The default ClassLoader can be extended to define specific properties for loading Java classes. Some of the properties that can be defined are:

- File Format (.class, .zip, .jar, .personal)
- Class Source (directories)
- Protocol (http, ftp, file)
- Security Conditions

Developers can create their own ClassLoaders by extending the ClassLoader abstract class. The ClassLoader abstract class contains the following methods. For a detailed explanation of the ClassLoader abstract class refer to "The Java Class Libraries An Annotated Reference" (Chan and Lee, Addison Wesley, 1997).

- `ClassLoader()` – This constructor must be instantiated by the calling application. The main purpose of this constructor, besides protocol-specific initialization, is to trigger a security check against the `checkCreateClassLoader` method in the acting `SecurityManager` instance. If a security violation exists, a `SecurityException` is thrown.
- `loadClass()` – This method is the only one whose implementation needs to be provided by a non-abstract ClassLoader.

`loadClass` method is responsible for mapping a class name to a class `Class` instance. The class loading policies are implemented inside of this method. The normal flow of execution of the `loadClass` method is to retrieve the class information in a byte array form, call the `defineClass` method to generate a `Class` object from the byte arrays, and to resolve the class references if the `resolve` variable is true. If the class name cannot be resolved to a class `Class` instance, a `ClassNotFoundException` is thrown.

- `defineClass()` – This method is used by the ClassLoader to generate a `Class` object from a byte array containing the byte codes for a specific class. Also, the `defineClass` method is responsible for introducing the generated class into the JVM runtime process. The byte array must contain the format of a valid class. If the byte array doesn't contain a valid class definition, a `ClassFormatError` is thrown.
- `findSystemClass()` – This method loads a class using the default system ClassLoader. This method uses the `java.class.path` system property to find the class that matches the requested class name. If the class name cannot be resolved to a class `Class` instance, a `ClassNotFoundException` is thrown.
- `resolveClass()` – This method is used by the ClassLoader to incorporate a class into the JVM runtime process. Another responsibility of the `resolveClass` method is to map all of the class references contained inside of the class being loaded (including super class, methods and fields).

The next section provides an example of a new ClassLoader, called the `URLClassLoader`, that extends the `ClassLoader` abstract class and leverages the Internet and Web browser to download Java classes into Java applications.

URLClassLoader

The `URLClassLoader` is a `ClassLoader` mechanism that leverages Web Servers to download Java classes to client Java applications. Java applets are able to download multiple Java classes from the connecting Web server. The `URLClassLoader` allows Java applications to similarly download



multiple Java classes, but unlike Java applets, from multiple Web servers. Once the classes are downloaded they are incorporated into the running Java application.

Unlike regular `ClassLoader` mechanisms which are file-based, the `URLClassLoader` requires a URL string in order to identify the location from where to download the class bytecodes. An example of a URL String is given below:

`http://www.motorservice.com/java/classes/texteditor.class.`

This String tells the ClassLoader to download a class call `texteditor.class` from the machine www.motorservice.com in the directory called `java_classes`.

The `URLClassLoader` is responsible for downloading the bytecodes from the URL location, instantiating a `Class` object from the downloaded bytecodes and caching the downloaded `Class` object for future use. These operations take place outside of the `loadClass()` method. The `loadClass()` method is called whenever an object is instantiated from the cached generated `Class` object using the `newInstance()` method on the `Class` class. We have chosen to download bytecodes and to process all of the class information outside of the `loadClass()` method because the URL information associated with the class name is not contained in the input parameters of the `loadClass` method. The caching policy may vary between `ClassLoader` implementations but the idea is the same, viz., to minimize the amount of time you go the network to access a class.

Applications of the URLClassLoader

The following types of applications can benefit from the capabilities of the URLClassLoader:

- Applications that need to download functionality dynamically based on user request
- Applications whose download time is very large and can benefit from sub-components download

- Applications that act as broker or trigger environments for classes without common interfaces
- Applications that need to interact with classes that do not share a common interface

The application environment presented in the next example is a trigger environment that starts Java applications. These applications are graphical in nature.

URLClass Loader Trigger Application

The trigger environment implemented in this example takes URL strings of the form

`http://www.motorservice.com/java/classes/texteditor.class`

The `Texteditor.class` is a Java application that extends the `Frame` class. The trigger application downloads this type of application over the Internet and executes them. This trigger environment acts as a runtime shell for Java programs. The advantage of having the trigger environment executing the various Java applications is that it can act as a task manager by suspending or stopping the execution of the spawned programs. The user could extend this program to allow components of one application to communicate and interact with components of a second application.

There are two classes associated with the trigger environment, trigger and URLClassLoader. Figure 1 shows the class diagrams of the two classes. The trigger class is responsible for providing the URLClassLoader with the desired Java application to be downloaded. Also, it is responsible for creating an instance of the downloaded Java application class and for starting its execution by calling the main() method on the class. The URLClassLoader is responsible for downloading the bytecodes associated with the URL string and for linking the downloaded Java application class with the running JVM process.

Figure 2 shows the collaboration diagrams for executing a Java Application downloaded over the Web.

The trigger environment provides a simple application that allows a user to enter a URL with a Java application class. In order for the Java application class to be downloadable it must be contained in a Web server directory. Figure 3 shows the trigger environment application screen.

The method `registerJavaClass()` creates a URL input stream and downloads the byte codes via this mechanism. This is the first method called whenever the "Execute Java Application" button is pressed. The string typed inside the textfield is passed to this method.

```
// Create URL Stream
u = new URL(URLlocation);

// Open URL Stream
input = u.openStream();
data = new DataInputStream(input);
```

The `registerJavaClass()` method continues by downloading the byte codes from the stream and creating a byte array.

```
// Download byte codes from URL Stream
byte classBytes[] = downloadByteCodesFromURL(data);
```

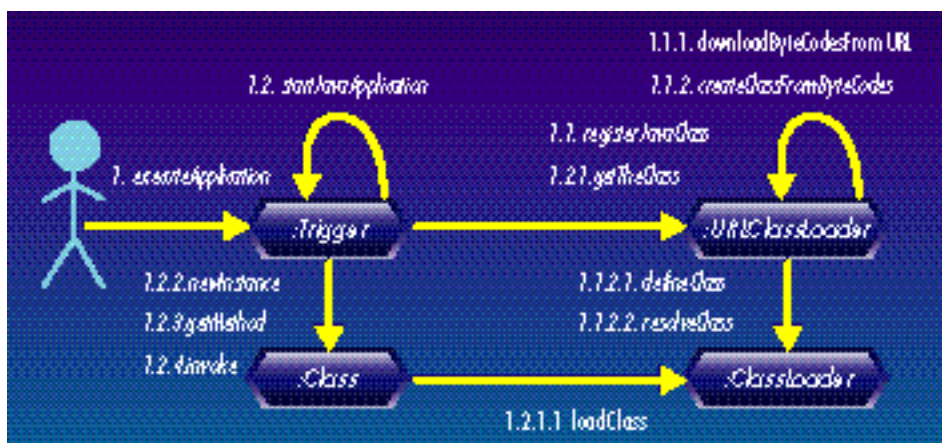


Figure 2: Collaboration diagrams



Figure 3: Trigger environment application screen

Also, the `registerJavaClass()` method creates a Class from the downloaded byte array.

```
// Create class from byte code
ourClass = createClassFromByteCodes(class-
Bytes, true);
```

The method `createClassFromByteCodes()` creates a class by calling the `defineClass()` method on the default ClassLoader. In addition, this method is responsible for linking the created class object with the JVM runtime process. Once the class object has been linked it caches the object inside of a Hashtable (see Listing 1).

The `loadClass()` method gets executed whenever the class is being instantiated. This method checks the local `java.class.path` system property first to

load the class locally. It checks the Hashtable cache if it can not find the class locally. If the class being instantiated has not been loaded it proceeds by trying to download the class from the last accessed Web site location (see Listing 2).

If the class being instantiated is not cached, the `URLClassLoader` attempts to download the requested class from the Web using the last accessed Web site. Fetching the class from the network is done by reconstructing a URL path to the requested class name (see Listing 3).

After the Java application class has been registered by the ClassLoader and instantiated by the trigger class, we use reflection to invoke the static main method inside the Java application. Since we are downloading Java applications over the Web, we expect the applications to contain a `main()`

method. We search the Java class for a `main(String[])` method and once we find it we execute the method (see Listing 4).

For a copy of the complete and working trigger program source code, please refer to Listing 5 and Listing 6. An additional enhancement that can be made to the trigger environment application is to persist the Hashtable object. This would allow the permanent caching of downloaded Java applications. However, the drawback to this approach is that as new releases of the applications are stored on Web servers you won't be able to use the new applications.

Conclusion

In this article, we walked through the design of a ClassLoader mechanism that allows you to use the Web servers as extensions of your hard drive. There are security issues involved in implementing your own ClassLoader that we have not discussed here. ☛

About the Author

Israel Hilerio is a member of the Technical Staff at i2 Technologies in Dallas, TX. He holds a B.S. in Computer Science Engineering from St. Mary's University and an M.S. in Computer Science from Southern Methodist University. Israel has eight years of programming experience, two of them in Java.



Israel_Hilerio@i2.com

Listing 1.

```
// Creates a class from the byte codes
c = defineClass(className, array, 0, array.length);
if (resolve) {

    // resolves all of the classes needed by the class
    // links the class into the JVM process
    resolveClass(c);

    // caches the class into a Hashtable
    classList.put(className, c);
}
```

Listing 2.

```
try {
    c = findSystemClass(name);
    System.out.println("Resolved " + name + " locally");
}

// If Class is Not Local, Attempt to Resolved it From Downloaded
Classes
catch (ClassNotFoundException e) {
    System.out.println("Resolving Class: " + name);
    classRequested = (Class) classList.get(name);

    if (classRequested == null) {
        System.out.println("Resolving " + name + " remotely");

        // If a Class has not been resolved previously then
        // attempt to load it from the last Web site that was
        // accessed.
        loadClassName( name );
```

```
c = ourClass;
    }
    }
    return c;
}
```

Listing 3.

```
public void loadClassName(String name) {
    String URLlocation;

    try {
        URLlocation = new String("http://" + serv + dir + "/" +
name + ".class");
        registerJavaClass(URLlocation);
    }
    catch (Exception e) {
        e.printStackTrace();
    }
}
```

Listing 4.

```
Object obj = c.newInstance();
...

// set the parameters for the static main method
Class array1[] = {Class.forName("Ljava.lang.String");};

// gets the static main method
m1 = c.getMethod("main", array1);

// invoke the static main method
m1.invoke(obj, object_array);
```


Listing 5: urlclassloader.java

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

class URLClassLoader extends ClassLoader {

    Class ourClass=null;
    Hashtable<Class> classList=null;
    int vector_counter;
    String directory=null;
    String server=null;
    String className=null;
    String UnresolvedURL=null;
    String temporaryURL=null;
    boolean flagClass;
    static int counter1;

    public URLClassLoader() {

        super();

        classList = new Hashtable();
    }

    public void registerJavaClass(String URLlocation) throws IOException {

        String filename=null;
        int slash_index=0;
        InputStream input=null;
        URL u=null;
        DataInputStream data=null;

        try {
            u = new URL(URLlocation);
            temporaryURL = new String(URLlocation);

            filename = new String(u.getFile());
            slash_index = filename.lastIndexOf("/");
            className =
                new String(filename.substring(slash_index+1,
                                                filename.length() - 6));

            directory =
                new String(filename.substring(0,
                                                slash_index));

            server = new String(u.getHost());

            input = u.openStream();

            data = new DataInputStream(input);

            byte classBytes[] =
                downloadByteCodesFromURL(data);

            data.close();
            input.close();

            input = null;
            u = null;
            System.gc();

            ourClass =
                createClassFromByteCodes(classBytes, true);

        } catch (Exception e) {

            if (input != null) {
                input.close();
                if (data != null) {
                    data.close();
                }
            }

            u = null;

            UnresolvedURL = new String(URLlocation);

            ourClass = null;

            throw new IOException("Problems Defining " +
                                   "Network Class +++++++>");
        }

        public byte[] downloadByteCodesFromURL(
            DataInputStream in) {

            ByteArrayOutputStream outputStream =
                new ByteArrayOutputStream();

            while (true) {
                try {
                    outputStream.write(in.readByte());
                }
                catch (IOException e) {
                    break;
                }
            }

            return outputStream.toByteArray();
        }

        public synchronized Class
        createClassFromByteCodes(byte[] array,
                                   boolean resolve) {

            Class c = null;

            try {
                c = defineClass(className, array,
                                0, array.length);

                if (resolve) {
                    resolveClass(c);
                    classList.put(className, c);
                }
            }
            catch (ClassFormatError e) {
                e.printStackTrace();
            }

            return (c);
        }

        public synchronized Class loadClass(String name,
                                                boolean resolve)
        throws ClassNotFoundException {

            Class c = null;
            Class classRequested;

            // Attempt to Load the Class Locally First
            try {
                c = findSystemClass(name);
                System.out.println("Resolved " + name +
                                    " locally");
            }
            // If Class is Not Local, Attempt to
            // Resolve it From Downloaded Classes
        }
    }
}
```

ADVERTISER INDEX

Intuitive
1/3 Page

Advertiser	Page	Advertiser	Page
3D Graphics www.threedgraphics.com	27 310 553-3313	Phaos www.Phaos.com	23 212 229-1450
Activeverse www.activeverse.com	21 512 708-1255	PreEmptive Solutions, Inc. www.preemptive.com	25 216 732-5895
Bristol Technology www.bristol.com	63 203 438-6969	Progress Software www.progress.com	29 617 280-4000
DCI www.DClexpo.com/Internet	68, 69 508 470-3880	ProtoView www.protoview.com	19 609 655-5000
Greenbrier & Russel www.gr.com/java	19 800 453-0347	Sales Vision www.salesvision.com	15 704 567-9111
Imperial www.Imperial.com	77 415 688-0200	SofTech Computer Systems www.scscompany.com	33 814 696-3715
Installshield www.installshield.com	31 800 250-2191	Software Development West www.sd98.com	57 800 411-8826
Intuitive www.intuitivesystems.com	50 408 245-8540	Stingray Software Inc. www.stingsoft.com	2 800 924-4223
JavaWorld www.javaworld.com	83 415 267-4527	SunTest www.suntest.com	67 415 336-2005
KL Group Inc. www.klg.com	84 800 663-4723	SuperCede www.asymetrix.com	6 800 448-6543
McGraw-Hill www.mcgrawhill.com	41 800 227-0900	Sybex Books www.sybex.com	37 510 523-8233
Marimba www.marimba.com/download	63 415 328-JAVA	Symantec cafe.symantec.com	3 800 453-1059 ext. 9NE5
Mecklermedia www.iworld.com	81 800 500-1959	SYS-CON Publications www.sys.con.com	61 914 735-1900
MindQ www.mindq.com	35 800 847-0904	SYS-CON Publications www.sys.con.com	78 914 735-1900
Net-Developer '98 www.net-developer.com	59 612 368-7227	SYS-CON Publications www.sys.con.com	79 914 735-1900
Net Guru www.ngt.com	67 800 know.ngt	Thought, Inc. www.thought.com	29 415 836-9199
Net Guru www.ngt.com	77 800 know.ngt	Waite Group Press www.waite.com	43 800 368-9369
Object Matter www.objectmatter.com	56 305 718-9109	Zero G. Software www.zerog.com	54 415 512-7771
Petronio Technology Group www.petronio.com	33 781 7788-2000		


```

        catch (ClassNotFoundException e) {
            System.out.println("Resolving Class: " +
                               name);

            classRequested =
                (Class) classList.get(name);

            if (classRequested == null) {
                System.out.println("Resolving " + name +
                                   " remotely");

                // If a Class has not been resolved
                // previously then attempt to load it
                // from the last web site that was accessed.
                loadClassName( name );
                c = ourClass;
            }
        }

        return c;
    }

    public void loadClassName(String name) {
        String URLlocation;

        try {
            URLlocation = new String("http://" + server
                                     + directory + "/" + name + ".class");
            registerJavaClass(URLlocation);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public Class getTheClass() {
        return (ourClass);
    }
}

```

Listing 6: trigger.java

```

import java.awt.*;
import java.net.*;
import java.io.*;
import java.lang.reflect.*;
import java.awt.event.*;
import java.util.*;
import URLClassLoader;

public class trigger extends Frame implements ActionListener{

    URLClassLoader loader=null;
    TextField tf = null;
    Button bt = null;

    public trigger() {
        super("Trigger Application");

        setLayout(new BorderLayout());
        tf = new TextField();
        bt = new Button("Execute Java Application");
        bt.addActionListener(this);

        add("North",
            new Label("Enter URL for loading Java application"));
        add("Center", tf);
        add("South", bt);

        loader = new URLClassLoader();

        setSize(300, 100);
    }

    public void actionPerformed(ActionEvent evt) {
        String URLlocation = tf.getText();

        if (URLlocation != null) {
            executeApplication(URLlocation);
        }

        public void executeApplication(
            String URLlocation) {

            try {
                loader.registerJavaClass(URLlocation);

                Class c = loader.getTheClass();

                startJavaApplication(c, "New Application");
            }
            catch (IOException e) {
                System.out.println("Error loading URL Class");
            }
        }

        public void startJavaApplication(
            Class c, String label) {

            try {

                Object obj = c.newInstance();

                String[] array = new String[2];
                array[0] = new String(label);
                Object object_array[] = {array};

                Method m1;

                // set the parameters for main method
                Class array1[] =
                    {Class.forName("[Ljava.lang.String;")};

                // get the static main method
                m1 = c.getMethod("main", array1);

                // invoke the static main method
                m1.invoke(obj, object_array);

            } catch (NoSuchMethodException e) {
                e.printStackTrace();
            } catch (ClassNotFoundException e) {
                e.printStackTrace();
            } catch (InvocationTargetException e) {
                e.printStackTrace();
            } catch (IllegalArgumentException e) {
                e.printStackTrace();
            } catch (IllegalAccessException e) {
                e.printStackTrace();
            } catch (InstantiationException e) {
                e.printStackTrace();
            } catch (NullPointerException e) {
                e.printStackTrace();
            }
        }

        public static void main(String args[]) {
            trigger application = new trigger();
            application.show();
        }
    }
}

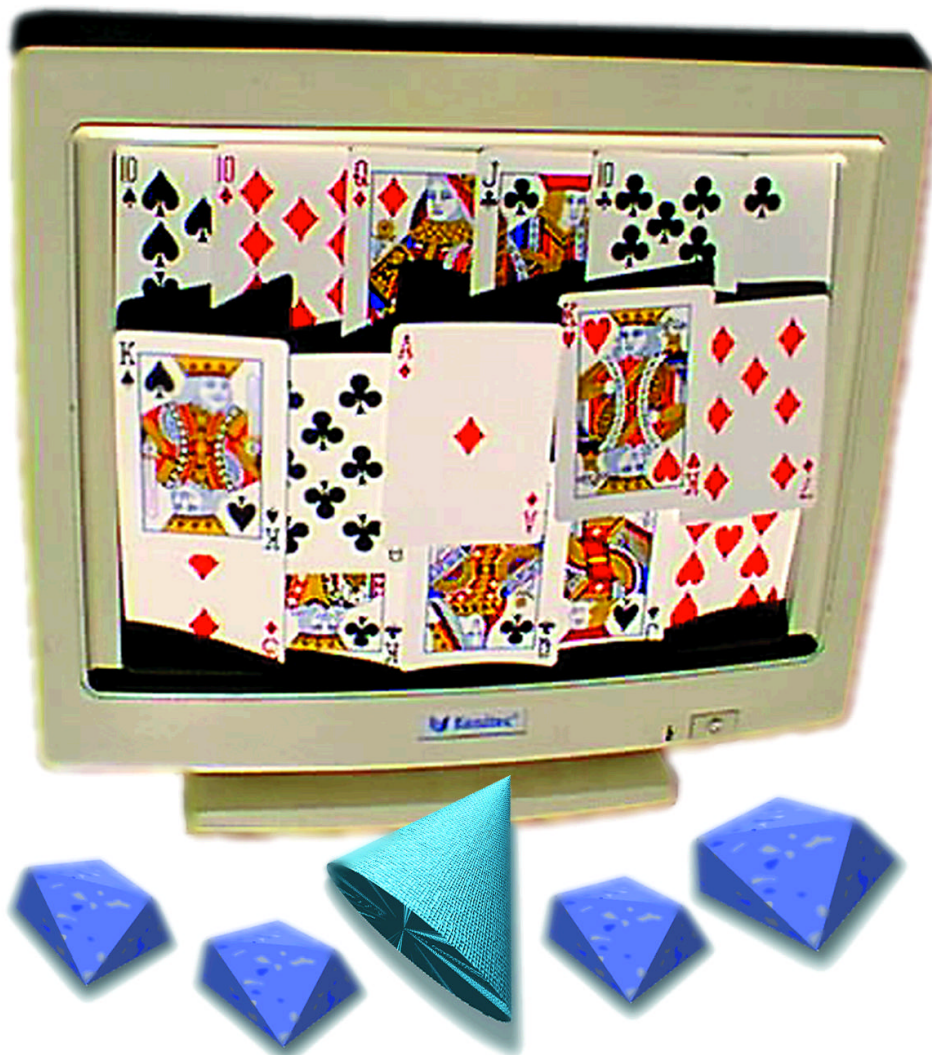
```

Developing C S Using Active Objects

PART 2

A useful paradigm for implementing collaborative client/server protocols

by Larry Chen & Todd Busby



Abstract

This is the second in a two-part series presenting a Java implementation of a real-time multi-user blackjack game based on a collaborative, active object framework. In the last article, we presented the design of an active object framework for developing collaborative client/server applications. In this article, we will use the active object framework to develop both the client and server components of a collaborative blackjack game.

Review of the Active Object Architecture

In last month's article, we presented an active object framework which supports event-based communication between concurrent objects. We defined an active object as a threaded object with an event queue capable of handling events from other active objects via both active waiting and passive callback handling. We also added support for client/server-based collaboration by defining the session concept, which is a group of active objects consisting of one server and one or more clients. We also simulated a multicast mechanism so that any active object within a session may send an event simultaneously to all other active objects within the session.

Simulating Client/Server Interaction

To demonstrate the use of our active object framework, we will keep the example simple by omitting some of the advanced functions of a blackjack game, such as betting, double-down and splitting. We will also not include any graphical interface in our simple example.

To begin, we define the blackjack server and blackjack client as active objects by extending `ActiveObject`.

```
class BJServer extends ActiveObject {
...
}
class BJClient extends ActiveObject {
...
}
```

Recall that our example active object architecture does not include networking capabilities. Therefore, for this example we will simulate multiple blackjack clients and one server all within one Java application as shown in Figure 1. This is accomplished by instantiating the server and all clients within a single application and calling each active object's `start()` methods. We will call this main application class `Blackjack`.

As shown in Listing 2b, the `Blackjack` application instantiates the `Blackjack` server and four `Blackjack` clients. The `Blackjack` server also represents the dealer, which we

will designate as seat number 0. The four clients represent four players occupying seats 1 through 4. Besides simplifying our example, this technique of running a group of interacting active objects within one application is a very useful technique to verify the correctness of a collaborative client/server protocol without the hassles of repeatedly restarting servers and clients during debugging.

Determining the Communication Protocol

The first step in creating a collaborative game such as blackjack is to define all of the events that will be necessary for communications between the clients and the server.

Following the active object design philosophy, the client and server will communicate through event passing between active objects. To design our blackjack client/server protocol, we will consider the

choices to hit or stand, so we need a `hit` and a `stand` event to reply to the server. During a sequence of hits, the player may bust, so the dealer needs to notify the client with a `you_bust` event. Finally, to conclude the game, the server needs to send a `game_result` event to notify the client that the current game is over.

In addition to control flow events, other events pertaining to the status of other players may occur during a game. These events will be handled by a separate entity, the passive callback handler, which is responsible for displaying information (or graphics) about all players during game play. In our simplified blackjack game, we only need two status events. First, the server needs to notify every client whenever a card is dealt to anyone, so that the card may be shown on the screen. This notification is represented by the `show_card` message. Note that the `show_card` message is used in addition to the `deal_card` control

Game Control Flow Events (Active)

<code>deal_card</code>	Sent by the server to inform a client of a card dealt to it.
<code>request_action</code>	Sent by the server to indicate that it's a particular client's turn.
<code>hit</code>	Sent by the client to inform the server of a player "hit" action.
<code>stand</code>	Sent by the client to inform the server of a player "stand" action.
<code>you_bust</code>	Sent by the server to inform a client that the player has bust.
<code>game_result</code>	Sent by the server to inform a client of the game's results.

Status Update Events (Passive)

<code>bust</code>	Sent by the server to inform a client that another player has bust.
<code>show_card</code>	Sent by the server to inform a client to display a card.

Table 1: Blackjack Events

mechanics of one session; that is, a client/server interaction between one server and up to four clients.

Recall that in our active object framework an active object may handle events from other active objects in two ways – using the active event handler or the passive callback handler. For blackjack we will use the active event handler to handle game control flow, and the passive callback handler to handle status update events (events that are used only for displaying status on the screen and has no effect on the game flow).

We'll first specify the events that are required to control one game (hand) of blackjack for a particular player. First, the dealer needs to deal two cards to the player, so let's define a `deal_card` event. Since this is a multi-player game, the player needs to wait until it is his turn to hit or stand. The dealer needs to notify the client with a `request_action` event. The player then

flow message because `show_card` is handled by the passive event handler while the `deal_card` message is handled by the active handler. We need another event to notify everyone that some player has bust. This is represented by the `bust` message. See Table 1 for a summary of the event definitions.

All events are defined by first defining a unique integer constant for each event type, which we define in class `Blackjack`:

```
public static final int hit = 1;
public static final int stand = 2;
public static final int game_result = 3;
...
```

Then, each event is defined as a subclass of `java.lang.Object`. Each event object implements `hashCode()` to return the unique integer identifier that we defined in class `Blackjack`. To define any parameters in the event, just define them as data members of the event class.

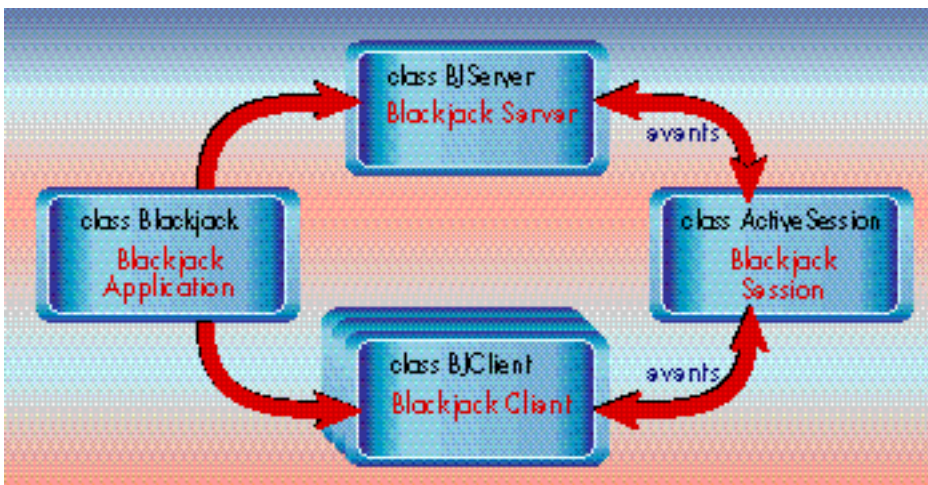


Figure 1: Client/server simulation using a single application

```

class deal_card extends Object
{
    int clientid;
    Card card;
    public deal_card(int clientid, Card card) {
        this.clientid = clientid;
        this.card=card;
    }
    public int hashCode() { return
    Blackjack.deal_card; }
}
  
```

The complete listing of event class definitions is given in Listing 2a.

We are now ready to code the server and client active objects, which will use the events we have defined to communicate.

Blackjack Server Logic

Using our active object framework, we can conceptualize a session as one blackjack table consisting of one dealer and up to four players. The blackjack session is identified by the session name "BJ." To allow for multiple concurrent blackjack tables, we can also assign each blackjack table a unique session ID. In our example here, the blackjack server is defined as class BJServer, extending ActiveObject. BJServer's constructor starts one session called "BJ" with a session ID of 0.

```

class BJServer extends ActiveObject
{
    private ActiveSession session;
  
```

```

    public BJServer()
    {
        this.session = new
        ActiveSession(this, "BJ", 0);
    }
    ...
  
```

Though an active object may handle events in both the run() or handleEvent() methods depending on the nature of the events, in this case the blackjack server only requires the run() method to control game flow. Note that blackjack server's run() method may generate status events directed toward the passive handler on the client (BJClient.handleEvent()) in addition to control flow events directed at the client active handler (BJClient.run()), as shown in Figure 2. BJServer.handleEvent() is not used and is empty.

As shown in Listing 2c, BJServer.run() is encased in an outermost while(true) loop so that the server may continuously serve games one after another. To begin a game, BJServer deals each player a card face up and a card face down.

```

for (int player=1;
    player<=Blackjack.MAX_PLAYERS; player++)
{
    deal CardTo(player, Card.FACE_DOWN);
    deal CardTo(player, Card.FACE_UP);
}
  
```

This is accomplished in the dealCard-

to() method by multicasting the status update event show_card to all clients and sending the deal_card to the client representing the player actually receiving the card.

```

    session.mcast(new
    show_card(player, card), 0); //show card to
    all except dealer
    session.send(new
    deal_card(player, card), player);
    //deal card to player
  
```

The dealCardTo() method also tracks the cards that each player has received so that the server may detect any player who has bust and the winner may be computed at the end of the game. See Listing 2c for details.

Next, it is required to serve each player. This involves sending a request_action event and waiting for a hit or stand event from the client. If a hit event is received, then a card is dealt and a check for a bust is performed. If a bust occurs, then a status event bust is multicast to all players using mcast() and a control event you_bust is sent to the player that has bust. This step is repeated until all of the players have properly been "served"; that is, until all players have requested to stand or have bust.

Finally, the server determines the winners and sends each player's game result using the event game_result.

Blackjack Client Logic

The Blackjack client also extends ActiveObject. Each Blackjack client is constructed with a clientID which corresponds to the player's seat number at the table. The first thing each client does is join the session, "BJ:0", which was started by the Blackjack server.

```

class BJClient extends ActiveObject
{
    protected int clientID;
    private ActiveSession session;

    public BJClient(int clientID)
    {
        this.clientID = clientID;
        this.session =
        ActiveSession.join(this, "BJ", 0);
    }
  
```

Listings 1a-1b: Active Object Framework
See Part 1 of this article in last month's issue.

Listing 2a: Blackjack Main

```

import java.awt.*;

public class Blackjack extends Object
{
    public static final int MAX_PLAYERS = 5;
  
```

```

    public static BJServer server;
    public static BJClient[] client = new BJClient[MAX_PLAYERS+1];

    public Blackjack()
    {
        server = new BJServer();
        for (int i=1; i<=MAX_PLAYERS; i++) client[i] = new BJClient(i);
    }
    public static void main(String[] args)
    {
  
```

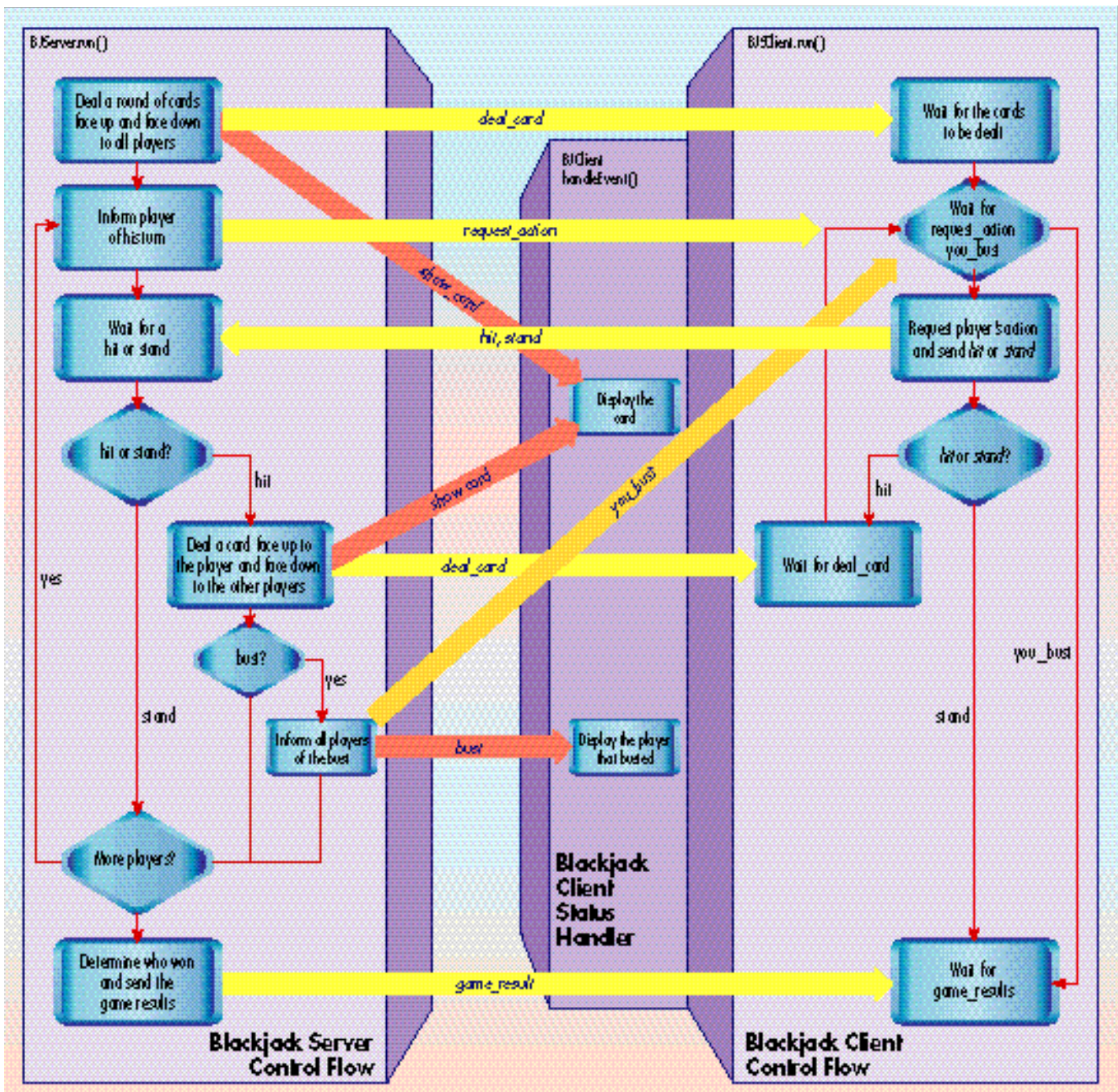


Figure 2: Blackjack client/server protocol and architecture

We also need to register all of the status update events that are to be handled by `handleEvent()`. This is done using `ActiveObject's registerCallback()` method.

```
int[] evts = {BlackJack.show_card, BlackJack.bust}; registerCallback(evts);
```

Both active control logic and status update logic events are required to be handled by the client; thus, both the `run()` method and `handleEvent()` method must be implemented.

Let's first look at the `handleEvent()`

method. There are only two status update events that we need to be concerned with for this example: `show_card` and `bust`. These can be handled by `handleEvent()` by checking the event's `hashCode()` in a switch statement, similar to this:

```
switch (evt.hashCode()) {
    case BlackJack.show_card: ...
    case BlackJack.bust: ...
}
```

In our example, we are just going to print out the status events that are received and their contents, since we are not implement-

ing a graphical interface.

Now let's implement the game control flow in `BJClient.run()`. Like `BJServer.run()`, we encase the `run()` method in an outermost `while(true)` loop so that multiple games may be played continuously. During a particular game, the first step is to wait for two cards to be dealt. This is done by performing two `waitFor()` methods, each like this:

```
waitFor(BlackJack.deal_card);
```

Then we need to wait for the control event, `request_action`, to start the client's turn. This is where the player is prompted

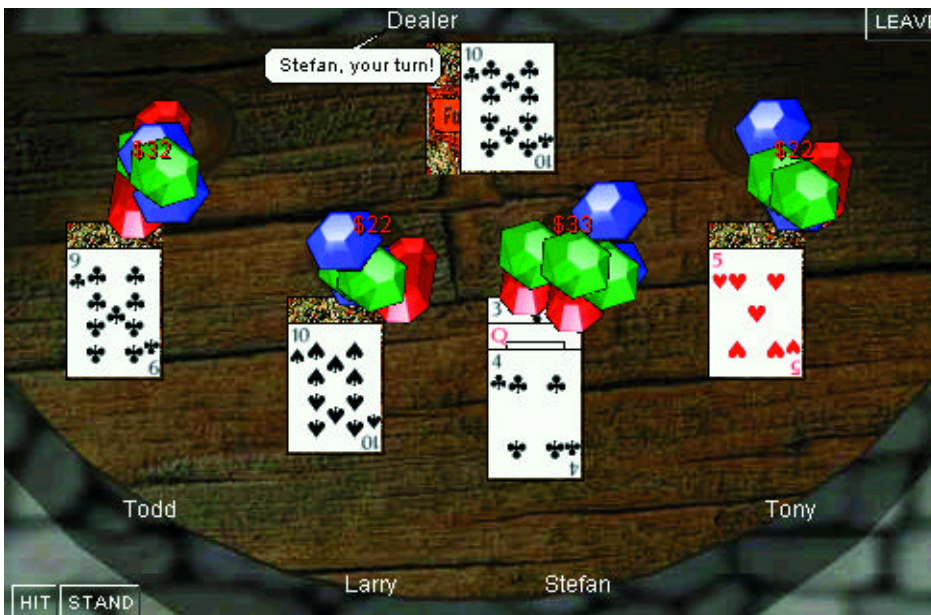


Figure 3: Funtopia Blackjack Game

whether he wants to hit or stand. A hit action by the player generates the control event `hit` and a stand action generates the control event `stand`. If the player hits then `BJClient` continues in a loop (which we call the `ActionLoop`), waiting for either `request_action` or `you_bust`. The client stays in the `ActionLoop` until either the player busts or the player requests to stand. In Listing 2d, we implement prompting for hit or stand by reading characters from the keyboard, so when a "stand" command is received, the thread needs to break out of the action loop. Since the action loop is not the immediate innermost loop at this point, we use

```
break ActionLoop;
```

to break out of the loop labeled with the `ActionLoop` label. Upon exiting the `ActionLoop`, all that's left to do is to wait for the control event `game_result`, which informs the client that the game is over and whether the player won or lost. The client then continues to wait for the start of the next game by restarting the outermost while loop.

Funtopia: A Real-World Implementation

In this two part article we have presented an active object framework suitable for

developing collaborative client/server applications and we have demonstrated use of this framework through an example multi-player blackjack game. Although major aspects such as networking and graphics were left out for simplicity, the active object design philosophy is one of the key concepts to successfully developing such collaborative applications.

Avanteer, Inc., a company dedicated to Java-based collaborative solutions, has developed a full-scale multi-player game site called Funtopia using the active object design philosophy. Funtopia, at <http://www.funtopia.com>, includes a fully-featured multi-player blackjack game (Figure 3) which includes the networking and graphics aspects of a collaborative game that we have omitted in this article. All of the source code listings given in this two part article is available on the Avanteer Web site, at <http://www.avanteer.com>. 🍀

About the Authors

Larry T. Chen is a co-founder of Avanteer, Inc., a company focusing on developing Java-based collaborative software. He is also pursuing a Ph.D. in the area of distributed systems at the University of California, Irvine. Larry may be reached at larryc@avanteer.com

Todd Busby is a project manager at Avanteer, Inc. Todd holds a Masters in Computer Science from the California State University, Fullerton. He may be reached at toddb@avanteer.com



```
new Blackjack();
server.start();
for (int i=1; i<=MAX_PLAYERS; i++) client[i].start();
}

public static final int hit = 1;
public static final int stand = 2;
public static final int game_result = 3;
public static final int request_action = 4;
public static final int bust = 5;
public static final int you_bust = 6;
public static final int deal_card = 7;
public static final int show_card = 8;
}
```

Listing 2b: Event Definitions

```
class deal_card extends Object
{
    int clientid;
    Card card;
    public deal_card(int clientid, Card card) {
        this.clientid = clientid;
        this.card=card;
    }
    public int hashCode() { return Blackjack.deal_card; }
    public String toString() {return "Your card "+card.toString();}
}
class show_card extends deal_card
```

```
{
    public show_card(int clientid, Card card) {super(clientid,
card);}
    public int hashCode() {return Blackjack.show_card;}
    public String toString() {return "Player "+clientid+" received
"+card.toString();}
}

class game_result extends Object
{
    int winner;
    public game_result(int winner) {this.winner = winner;}
    public int hashCode() {return Blackjack.game_result;}
    public String toString() {return "Player "+winner;}
}
```

Listing 2c: Blackjack Server

```
class BJServer extends ActiveObject
{
    private ActiveSession session;
    private int[] cardTotals = new int[Blackjack.MAX_PLAYERS+1];
    private int[] aceCount = new int[Blackjack.MAX_PLAYERS+1];

    public BJServer()
    {
        this.session = new ActiveSession(this, "BJ", 0);
    }

    public void handleEvent(Object evt) {}
}
```


Software Developer full

```

public void run() {
    while (true)
    {
        //deal cards
        for (int player=1; player<=Blackjack.MAX_PLAYERS; player++)
        {
            dealCardTo(player, Card.FACE_DOWN);
            dealCardTo(player, Card.FACE_UP);
        }

        //serve each player
        for (int player=1; player<=Blackjack.MAX_PLAYERS; player++)
        {
            Object evt = null;
            do
            {
                session.send(new request_action(), player);

                int[] evts = {Blackjack.hit, Blackjack.stand};
                evt = waitFor(evts);
                if (evt.hashCode()==Blackjack.hit)
                {
                    dealCardTo(player, Card.FACE_UP);
                    if (cardTotals[player] > 21)
                    {
                        cardTotals[player] = 0;
                        session.mcast(new bust(player), 0);
                        session.send(new you_bust(), player);
                        break; //breaks out of do loop
                    }
                }
            } while (evt.hashCode()!=Blackjack.hit);
        }

        //send results
        session.mcast(new game_result(winner()), 0);
    }

    void dealCardTo(int player, int face)
    {
        Card card = Deck.dealCard(face);
        if (card.getValue()==1) aceCount[player]++;
        cardTotals[player] += (card.getValue()==1 ? 11 : card.getValue());

        while (cardTotals[player] > 21 && aceCount[player] > 0)
            if (cardTotals[player] > 21) { cardTotals[player]-=10; aceCount[player]--; }

        session.mcast(new show_card(player, card), 0);
        session.send(new deal_card(player, card), player);
    }

    int winner()
    {
        int max = 0;
        int theWinner = 0;
        for (int i=1; i<=Blackjack.MAX_PLAYERS; i++)
            if (cardTotals[i] > max) {max = cardTotals[i]; theWinner = i;}
        return theWinner;
    }
}

```

Listing 2d: Blackjack Client

```

class BJClient extends ActiveObject
{
    protected int clientID;

    private ActiveSession session;
    private TextArea textBox;

    public BJClient(int clientID)
    {
        this.clientID = clientID;
    }
}

```

```

this.session = ActiveSession.join(this, "BJ", 0);
Frame f = new Frame(this.getClass()+" "+clientID);
f.add("Center", textBox = new TextArea(20, 40));
f.pack(); f.show();
int[] evts = {Blackjack.show_card, Blackjack.bust};
registerCallback(evts);
}

```

```

public void handleEvent(Object evt)
{
    switch (evt.hashCode())
    {
        case Blackjack.show_card:
            print(" "+evt);
            break;
        case Blackjack.bust:
            print(" "+evt);
    }
}

```

```

public void print(Object o)
{textBox.appendText(o.toString()+"\n");}

```

```

public void run()
{
    while(true)
    {
        print(waitFor(Blackjack.deal_card));
        print(waitFor(Blackjack.deal_card));
    }
}

ActionLoop:
while (true)
{
    int[] evts = {Blackjack.request_action, Blackjack.you_bust};
    Object evt = waitFor(evts);
    if (evt.hashCode() == Blackjack.you_bust)
    {
        print("You bust!");
        break;
    }

    print("[H]it or [S]tand?\n");
    char c = 's';
    do { try {c = (char) System.in.read();}
        catch (Exception e) {e.printStackTrace();}
    } while (c != 'h' && c != 'H');
    session.send(new hit(), 0);
    print(waitFor(Blackjack.deal_card));
    break; case 'S': case 's':
        session.send(new stand(), 0);
        break ActionLoop;
    default: } while (c != 'h' && c != 'H');
} print("The winner is "+waitFor(Blackjack.game_result)+"\n-----"); } }

```

```

class bust extends Object
{
    int player;
    public bust(int player) {this.player = player;}
    public int hashCode() {return Blackjack.bust;}
    public String toString() {return "Player "+player+" busts!";}
}

```

```

class request_action extends Object
{public int hashCode() {return Blackjack.request_action;}}
class hit extends Object
{public int hashCode() {return Blackjack.hit;}}
class stand extends Object
{public int hashCode() {return Blackjack.stand;}}
class you_bust extends Object
{public int hashCode() {return Blackjack.you_bust;}}

```

Net Developer full

Java & CORBA

How close are they really?

by Gerald Brose

While it is easier to write CORBA applications in Java than in any other language, there are still a few conceptual stumbling stones that Java/CORBA programmers need to be aware of.

The combination of Java and CORBA, the OMG's middleware standard, has been receiving a lot of interest in both the Java and CORBA communities because of their apparent similarities in syntax and concepts. An OMG-defined language mapping and a number of existing commercial and public domain CORBA platforms open up the CORBA world for Java applications and components. In this article, I look at how far the integration of Java and CORBA can really go and point out a few pitfalls that arise because of subtle and not-so-subtle mismatches between the two worlds.

Introduction

In the Java/CORBA context, the term "integration" occurs frequently as a synonym for the combination of Java and CORBA technology. What this term conveys nicely is that the combination of these technologies promises more than just the sum of its parts as, e.g., in "language A plus thread library Y". With the conceptual distance between Java and CORBA so small, you can actually expect CORBA and Java to integrate smoothly, with only a few additional aspects that need to be understood by Java developers.

Java/CORBA integration can be achieved at two different levels: Level one is where CORBA concepts as expressed in IDL – the CORBA Interface Definition Language – are mapped to Java so that Java developers can now use them. This is what we have today: a standardized IDL-to-Java language mapping and IDL compilers from different vendors which realize it. However, this views the world from the CORBA perspective only: You design your applications in IDL using concepts not present in Java

and implement them by modelling these features with Java language constructs.

At the second level, however, you can design distributed applications using a homogeneous Java programming paradigm rather than IDL. An easy-to-use distribution platform, something like a CORBA-enhanced RMI, makes your Java objects or components accessible from a CORBA-environment – without IDL specifications. To achieve this degree of integration, it must be possible to easily map both ways between Java and CORBA, so there is a need for a reverse mapping from Java to IDL in addition to the already standardized IDL-to-Java mapping. Existing approaches like Visigenic's Caffeine point in that direction, and the OMG is actively working on a Java-to-IDL mapping.

A few weak spots remain at both levels of integration. In the following sections, I'd like to go into more detail and explain why 100 percent integration is not possible.

Mapping the CORBA Object Model

Let's look at how the CORBA object model is mapped to Java. Most of this mapping is fairly straightforward, but some IDL concepts map less easily than others and need extra consideration.

The CORBA object model is defined by the concepts of IDL. Since CORBA addresses implementation only in the individual language mappings, IDL does not have a notion of classes – only interfaces (object types) can be defined. Interfaces can inherit from other interfaces and define operations and attributes. Operations can throw exceptions as in Java, but unlike in Java, operation names may not be overloaded. Also, the parameter passing modes are more varied than in Java. Apart from the fact that object-type arguments are always passed by reference and base-type arguments are passed by value, the direction in which arguments are passed has to be

Like many technologies with openly specified standard definitions, CORBA is represented in the marketplace not only by commercial implementations but also by a number of free implementations. The column author for this issue wrote one of these implementations, JacORB, completely in Java, and here reflects on the matches and mismatches between Java and CORBA today.

Richard Soley
Editor, CORBACORNER
President and Technical Director of the
Object Management Group, Inc.

selected to be one of in, out or inout. While in parameters are passed to the called context, out parameters are passed back to the calling context when an invocation returns. The inout mode is a combination of the other two modes. Listing 1 is an example of an IDL specification.

Mapping the specification from Listing 1 to Java requires mapping some concepts not present in Java; in this case typedefs, sequences, bounded arrays and out parameters. The output the IDL compiler produces for the two IDL interfaces is shown in Listing 2.

As you can see, the IDL interfaces map directly to Java interfaces which also preserve the inheritance relationship between writer and server. Also note that the typedef'd names `stringArray` and `longs` do not appear in the mapped Java code. Typedef'd names can generally be replaced by the name of the original type they denote, as the mapping did for `longs`. In the case of `stringArray` however, it would not be sufficient simply to replace it with `String[]`. First, Java does not permit declaring array types with fixed bounds as in the IDL file above, so these kinds of array declarations result in extra generated Java classes which know about their array bounds. The IDL compiler therefore generates a class `StringArray5`. However, we're not done yet; the IDL compiler has to generate another class to model the out parameter of the

New House ad full

array operation: `StringArray5OutHolder`. This class finally is the one used for the parameter type in the Java interface.

While mapping IDL as sketched above can be standardized and taken care of by tools, a few object model mismatches between Java and CORBA remain and deserve mention.

Object Model Mismatches

First, Java has no unsigned integers as CORBA does. This is not a big problem, but it is annoying that programmers have to take care that large positive integer values, e.g., results from operations on other CORBA objects, do not exceed the range of positive values of Java integers.

Another, more serious, point is that strings and arrays are object types in Java whereas they are base types in CORBA. This has practical implications: First, you cannot pass a null reference to either a string or an array as an argument to a CORBA operation. As these types are no reference types in CORBA, only empty strings and arrays will be accepted:

```
Server s = Somewhere.getServer();
s.write(""); // ok
String str = null;
s.write( str ); // illegal
```

Second, and this is even more problematic, operations with array parameters might exhibit different behaviors depending on the kind of object reference used to call them and thus on the parameter passing semantics employed. In the case of

strings, this is not problematic: Strings are immutable, so operations on a string from within an operation's context are not visible in the calling context. With arrays, however, there is a real problem.

Consider the implementation of the operation `sum` in Listing 3: `Sum` calculates the sum of two integer arrays of equal length by adding the items at each index position and returns an array containing the results. If invoked as a CORBA operation, both `a` and `b` are passed by value and so is the result. For efficiency reasons, the method does not create another array for the results but rather reuses `a`. Everything works fine as long as we are in the world of CORBA semantics and all arguments are passed by value. If, however, `sum` is called using a Java object reference, its arguments are passed by reference and the actual argument supplied for `a` will have its contents overridden by the result.

Ultimately, what this example shows is that there can be no access-transparent integration of Java and CORBA in any platform: It must be clear at all times which invocation semantics apply in order to avoid unpleasant surprises. If the designer of `sum` intended it to be used with CORBA semantics only, users must take care that they never access objects of this class using local Java references!

Actually, this is not an inherent problem of CORBA. Rather, it is due to the fact that the Java language design defines parameter

passing modes which cannot be applied consistently in both local or remote calls. The Java approach of passing arrays by reference is not appropriate for remote invocations, so any distribution platform worth its salt will pass arrays by value in remote invocations. Consequently, this problem

“...there is a need for a reverse mapping from Java to IDL in addition to the already standardized IDL-to-Java mapping.”

will surface in any Java platform for distributed programming; e.g., in RMI [2].

Method Overloading, Classes, Object Serialization

In the previous sections the focus was on CORBA concepts without exact equivalents in Java. Let's now examine three Java concepts not present in CORBA – method overloading, object serialization and classes.

While overloading method names is common in Java, it is not allowed in IDL. Fortunately, this need not pose problems even when making existing Java applications CORBA-aware – if you have a tool providing a reverse mapping from Java to IDL. In the process of reverse mapping, this tool will simply rename overloaded names apart; i.e., mangle them. If you only design in IDL, there will simply be no overloaded names in your Java implementations.

Another concept frequently considered missing in CORBA is the option to pass objects by value, also known as object serialization from RMI. In many situations, such as when an object is actually not a full-blown ADT which encapsulates state but a simple container of data, it is much more appropriate to copy this object to the operation than to pass a reference to it: i.e., pass it by value. The OMG has acknowledged the need for passing objects by value and has issued a corresponding request for proposals, so the process to enhance IDL with this concept is under way.

If we now look at the class concept of Java, we find that it has three different aspects. Classes define object types, they represent object implementations and also serve as a template for object creation. The

Listing 1: Example IDL specification.

```
module example
{
    typedef string stringArray[5];
    typedef sequence<long> longs;

    exception UnequalLength {};

    interface writer {
        void write(in string s);
    };

    interface server: writer
    {
        void arrayfy(in string s,
                     out stringArray);
        longs sum(in longs a,
                  in longs b)
            raises (UnequalLength);
    };
};
```

Listing 2: Output produced by IDL compiler.

```
package example;
public interface writer
```

```
{
    void write(String s);
}

package example;
public interface server
    extends example.writer
{
    void arrayfy( String s,
                  StringArray5OutHolder sa );

    int[] sum(int[] a, int[] b)
        throws UnequalLength;
}
```

Listing 3: Implementation of the operation `sum`.

```
int[] sum(int[] a, int[] b)
    throws UnequalLength
{
    if( a.length != b.length )
        throw new UnequalLength();
    for(int i=0; i<a.length; i++)
        a[i] = a[i] + b[i];
    return a;
}
```


aspect of object implementation is obviously beyond the scope of CORBA, but what about types and object creation?

As already mentioned, object types can only be defined with interfaces in IDL. It is, however, not a problem to use classes as object types in Java, even if you plan to distribute your Java application with CORBA later. A decent reverse mapping is quite capable of implicitly or explicitly deriving IDL interfaces and generating stubs and skeletons from Java classes.

What about instantiating objects now? In a normal Java program you simply call a class's constructor and the runtime system will provide you with a reference to a newly created object, so you need only assign the resulting object reference to a variable.

```
X my_x = new XImpl();
```

Implicit in this way of creating and binding objects are two things that cannot be handled implicitly in a distributed environment. First, you have not only created an object of a particular type X, but also had to know about a specific implementation, XImpl. Second, there is no way to influence the actual location at which the object was created. Both these restrictions are not generally acceptable for creating objects in open distributed systems, so creating objects has to be done in a different fashion.

As you cannot rely on classes for creating object instances in CORBA, you have to use special objects which explicitly list a creation function for other objects in their interface. These "creator" objects are called factory objects or factories. Using factories to create objects, you need not care about which implementation of the desired object type is actually used – that's determined by the factory's implementation. If you really need to know, you can always provide a specific factory implementation yourself. To control object locations, CORBA defines a dedicated service, the Life Cycle Service, which provides means to create objects at specific locations or, if applicable, move them around. However, this is not a language construct of IDL and thus not an integral part of the CORBA object model – which in itself is location transparent. Also, not all CORBA platforms actually come with this service.

Summary

In this article I have outlined a few mismatches between Java and CORBA and pointed out their practical implications. Most of them are not critical as such and can be worked around with either tool support or by following certain programming conventions.

But wouldn't it be nice if Java eventually

came with proper parameter passing modes and perhaps a few other modifications so that it could really be called a CORBA implementation language? A language with just the right concepts and constructs to make implementing CORBA objects as easy as Java programming is today?

References

1. The OMG documents on the IDL-to-Java and the Java-to-IDL mappings are at http://www.omg.org/library/schedule/IDL_to_Java_RFP.html and [- a_to_IDL_RFP.ntml respectively
 2. Gerald Brose, Klaus-Peter Löhr, Andre Spiegel, "Java Does not Distribute", Procs. TOOLS Pacific 97, Melbourne, Australia. ☛](http://www.omg.org/library/schedule/Jav</div><div data-bbox=)

About the Author

Gerald Brose is a research assistant and PhD candidate at Freie Universität Berlin, Germany, and the author of JacORB, a free Java implementation of CORBA. He can be reached through e-mail at brose@inf.fu-berlin.de.



brose@inf.fu-berlin.de



News Today, Gone Tomorrow

Building a simple class to contact Usenet servers

by Alan Williamson

From the very first steps this column took, a journey of discovery was promised which, hopefully, has not disappointed. Using Visual Café as our development platform, we have ventured into areas of Java that others have feared to tread. From building complete applets with database connectivity to developing classes for sending and receiving e-mail, this column has so far tackled a wide variety of different problems. On the face of it, many of the topics appear difficult or far too complicated to have in an ordinary applet, application or servlet. But I think you'll agree that once the initial wall has been scaled, the rest of the problem becomes somewhat academic.

My previous two columns looked at interfacing to two of the more popular TCP services; POP and SMTP. This month, we will round off the TCP services with an implementation for a generic class to interface to the Usenet newsgroup network.

Usenet

The Usenet, or newsgroups, is a place where users exchange information in topics of conversation. These range from computing right through to knitting. In order to view or posts news items, a Usenet viewer program is required. This is simply a program that interfaces to a Usenet server and retrieves and post news articles. One of the more popular Usenet clients for the PC platform is Free Agent. Free Agent gives an interface to the newsgroups, downloading and uploading appropriate articles. It is this functionality we will build into a class that will allow any Java program to have access to the Usenet.

Connecting to a Usenet server is a simple matter of opening up a socket connection to the server which usually sits and listens for client connections on port 119. The procedure for performing this can be seen in the complete listing of the class. Like the POP

and SMTP servers, communication with the Usenet server is through command strings passed as carriage-returned lines.

The available command set to communicate with the server is surprisingly small, with only a handful of commands required to get the most out of a server. But before we go into the commands, let's take a quick look at how articles are arranged on the server.

Articles

Each article that exists on the Usenet system is identified with a global message id. This is normally created using a mixture of time/date/posters domain, to ensure complete uniqueness across all the newsgroup servers. When a new article is posted to a server, the server then distributes the article throughout the network. This procedure propagates the article throughout the whole network.

Associated with each article is a header. This header, shown in Listing 1, describes the article, which includes the e-mail of the poster, the subject of the article, the newsgroup the article has been posted in and the date it was posted (to name some of the more useful fields).

Retrieving the article from the server is a simple matter of issuing the commands

```
HEAD <message-id>
BODY <message-id>
```

where <message-id> is the ID of the article, as stated in the Message-ID field of the article header. If the article exists, it will be sent back to the client. The header is sent first, then the body. With this, we can write a simple method to pull a particular article back from the server. Listing 2 illustrates this method.

Listing 2 shows the method, `getArticle(...)` which downloads the article from the server. First, it issues the HEAD command which asks the server to download the head-

er of the given article. As each line is received, it is parsed so we may extract the e-mail address of the poster, the date it was posted and the subject of the article. If for some reason, the article is no longer available, then the server will return an error message line which contains a status code at the start of the line. If this code is either 423 or 430, we can assume the article is no longer available. If this is the case, the body is not downloaded and the method returns a null.

The body operates on somewhat the same principle, copying the lines of the article into a Vector for later use. Note that in this version we are actively filtering out all binary attachments as this can make the article body very large indeed. You may have noticed that the method returns a class of type `cArticle`. This is a simple wrapper class that represents a single Usenet article, with the body held in a Vector class.

Posting Articles

Posting an article to the Usenet server is much easier than downloading one. Using the command POST, the server will expect the head and then the body of the article, terminated with a single period (.) on a line of its own. The head is separated from the body by a single blank line.

Listing 3 shows the implementation of the `postArticle(...)` method. This method takes in the newsgroup the article is to be posted to, the e-mail of the poster, the subject and finally the actual body of the article formatted in a Vector, where each element is a single line.

The most important part of posting to the server is correctly formatting the header. If any of the syntax is wrong, then the article will be rejected. One of the features of the Usenet system is the ability to cross-post. This is where the same article is posted to multiple newsgroups at the same time. This is particularly handy as it saves on both bandwidth and time. To do this, simply list the newsgroups as a comma separated list. The server will then do the rest without any further intervention from the client.

Article Listings

So far we have seen how to post an arti-

cle to the newsgroups, and now to request a particular article based on its message identification. However, this begs the question, how does one determine the message ids of the articles, so the client optionally may download them?

Fortunately, this is a fairly easy operation to perform. The server can be asked to return all the message ids for a particular newsgroup, since a certain date. For example, to request all the messages since the 9th of December 1997, midnight, for the newsgroup comp.lang.java.programmer, the following command would be issued

NEWSNEWS comp.lang.java.programmer 971209
000000

The server will then return a list of message ids, each in their own line, with the list terminated using a period (.). This message id can then be used in the method described

earlier to retrieve that article. Listing 4 demonstrates the communication with the server to determine the list of message ids.

The method returns a Vector of message ids back to the caller, or null if an error occurred. Again, as with the posting of articles, the formatting of the date is most important; otherwise, the server will recheck the command.

Summary

This article looked at the NNTP (News Network Transport Protocol) protocol, which is used to communicate with the Usenet servers. By building a simple class, the majority of the functionality required to send and receive articles is a simple method call away. Although only a subset of the available commands was implemented, other commands exist that allow further interrogation of the server. For example, a command exists that determines the available newsgroups.

Communication with the more popular TCP servers is performed using simple ASCII commands, which allow for very straightforward clients to be implemented. This is one of the real powers the Internet has to offer.

In the next column we will go further down the road of Java enlightenment and tackle another 'looks-far-too-complicated' topic area. ☛

About the Author

Alan Williamson is on the Board of Directors at N-ARY Limited, a UK-based Java software company specializing in Java/JDBC/Servlets. He has recently completed his second book, focusing purely on Java Servlets. His first book looked at using Java/JDBC/Servlets to provide a very efficient database solution. He can be reached at alan@n-ary.com (<http://www.n-ary.com>) and he welcomes all suggestions and comments.



alan@n-ary.com

Listing 1: Header.

```
Path: news.wisper.net!peer.news.zetnet.net!di.spose.news.demon.net!
From: alan@n-ary.com (AR Williamson)
Newsgroups:
comp.lang.java.programmer, comp.lang.java.help, comp.lang.java.gui
Subject: Java Servlets
Date: 28 Nov 1997 13:23:02 -0800
Organization: N-ARY Limited
Message-ID: <65ncnm$qq$1@n-ary.com>
NNTP-Posting-Host: n-ary.com
X-Trace: 880752184 11160 pvd1 206.184.139.132
Lines: 31
```

Listing 2: getArticle(...) method.

```
public cArticle getArticle( String ID ) throws IOException
{
    cArticle NG = new cArticle();
    String LineIn;
    boolean bBad = false;

    //- Get the Header
    sendMessage( "HEAD " + ID );
    while ( (LineIn=In.readLine()) != null )
    {
        if ( LineIn.charAt(0) == '.' )
            break;
        else if ( LineIn.indexOf("423") == 0 || LineIn.indexOf("430")
== 0 )
            return null;
        else if ( LineIn.indexOf("Subject:") != -1 )
            NG.Subject = LineIn.substring( LineIn.indexOf("Sub-
ject:") + 9, LineIn.length() );
        else if ( LineIn.indexOf("Date:") != -1 ) {
            try{
                NG.EntryDate = new Date( LineIn.substring( LineIn.index-
Of("Date:") + 6,
                    LineIn.length() ) ).getTime();
            }
            catch( Exception E ){
                NG.EntryDate = new Date().getTime();
            }
        }
        else if ( LineIn.indexOf("From:") != -1 ){
            NG.Author = LineIn.substring(
LineIn.indexOf("From:") + 6, LineIn.length() );
```

```
NG.Author = NG.Author.replace( ' ', ' ' );
    }
    else if ( LineIn.indexOf("Lines:") != -1 ){
        try{
            int Lines = Integer.parseInt(
LineIn.substring( LineIn.indexOf("Lines:") + 7, LineIn.length() ) );
            if ( Lines > 120 )
                bBad = true;
        } catch( Exception E ){
        }
    }

    //- Get the Body
    sendMessage( "BODY " + ID );
    NG.vBody = new Vector(30, 5);
    LineIn = In.readLine();
    if ( LineIn == null || LineIn.indexOf("222") == -1 ) return
null;

    while ( (LineIn=In.readLine()) != null )
    {
        if ( LineIn.length() != 0 && LineIn.charAt(0) == '.' &&
LineIn.length() == 1 )
            break;
        else if ( LineIn.length() != 0 && LineIn.indexOf("Cont ent-
type:") != -1 ){
            if ( LineIn.indexOf("text/plain") != -1 )
                NG.vBody.addElement( LineIn + "\n" );
            else
                bBad = true;
        }
        else if ( LineIn.length() != 0 && LineIn.indexOf("begin 600")
!= -1 )
            bBad = true;
        else if ( bBad == false )
            NG.vBody.addElement( LineIn + "\n" );
    }

    if ( bBad ) return null;
    NG.Newsgroup = Location + Type + cArticle.USENET;
    return NG;
}
```

Listing 3: postArticle(...) method.

```
public void postArticle( String _NG, String _from, String _Sub-
ject, Vector _Body ) throws IOException
```

```

{
    String LineIn;

    //- Get the Header
    sendMessage( "POST" );
    LineIn = In.readLine();
    if ( LineIn == null || LineIn.indexOf("340") == -1 ) return;

    //- Post the Header fields
    sendMessage( "Path: n-ary" );
    sendMessage( "From: " + _from );
    sendMessage( "Newsgroups: " + _NG );
    sendMessage( "Subject: " + _Subject );
    sendMessage( "Message-ID: <" + System.currentTimeMillis() + "@n-ary.com>" );
    sendMessage( "Date: " + new SimpleDateFormat( "dd MMM yy HH:mm:ss").format(new Date()) + " GMT");
    sendMessage( "Organization: N-ARY Limited" );
    sendMessage( "Reply-To: " + _from + "\r\n" );

    //- Post the Body
    Enumeration E = _Body.elements();
    while ( E.hasMoreElements() )
        sendMessage( (String)E.nextElement() );

    sendMessage( "\r\n.\r\n" );
    In.readLine();
}

```

Listing 4.

```

public Vector getArticleList(String _Name, long _date) throws
IOException
{
    String D = new SimpleDateFormat( "yyMMdd HHmmss").format(new
Date(_date));

    Vector V = new Vector( 50, 10 );
    sendMessage( "NEWNEWS " + _Name + " " + D );
    String LineIn;
    while ( (LineIn=In.readLine()) != null ){
        if ( LineIn.charAt(0) == '.' )
            return V;
        else if ( LineIn.indexOf( "230" ) == 0 )
            continue;
        else
            V.addElement( LineIn );
    }
    return null;
}

```

Listing 5: Complete listing.

```

import java.applet.*;
import java.io.*;
import java.util.*;
import java.net.*;
import java.text.*;

public class newscomms
{
    DataInputStream In;
    DataOutputStream Out;
    Socket OutPort;

    public newscomms(){}

    public boolean openPort(String _Host){
        try{
            OutPort = new Socket( _Host, 119 );
            In = new DataInputStream( OutPort.getInputStream()
);
            Out = new DataOutputStream( OutPort.getOutputStream() );
            In.readLine();

```

```

            return true;
        }catch( IOException E ){
            System.out.println( "newscomms.openPort(): " + E );
        }
        return false;
    }

    public void closePort(){
        try{
            sendMessage( "QUIT" );
            OutPort.close();
        }catch( IOException E){}
    }

    public void sendMessage( String _M ) throws IOException{
        Out.writeBytes( _M + "\r\n" );
    }

    public Vector getArticleList(String _Name, long _date) throws
IOException{
        String D = new SimpleDateFormat( "yyMMdd HHmmss").format(new
Date(_date));

        Vector V = new Vector( 50, 10 );
        sendMessage( "NEWNEWS " + _Name + " " + D );
        String LineIn;
        while ( (LineIn=In.readLine()) != null ) {
            if ( LineIn.charAt(0) == '.' )
                return V;
            else if ( LineIn.indexOf( "230" ) == 0 )
                continue;
            else
                V.addElement( LineIn );
        }
        return null;
    }

    public cArticle getArticle( String ID ) throws IOException {
        cArticle NG = new cArticle();
        String LineIn;
        boolean bBad = false;

        //- Default the Newsgroup
        int Location = cArticle.UK;
        int Type = cArticle.PERMANENT;

        //- Get the Header
        sendMessage( "HEAD " + ID );
        while ( (LineIn=In.readLine()) != null )
        {
            if ( LineIn.charAt(0) == '.' )
                break;
            else if ( LineIn.indexOf("423") == 0 || LineIn.index-
Of("430") == 0 )
                return null;
            else if ( LineIn.indexOf("Subject:") != -1 )
                NG.Subject = LineIn.substring( LineIn.indexOf("Sub-
ject:")+9, LineIn.length() );
            else if ( LineIn.indexOf("Date:") != -1 ) {
                try{
                    NG.EntryDate=new Date(LineIn.substring(LineIn.index-
Of("Date:")+6, LineIn.length())).getTime();
                }catch( Exception E ){
                    NG.EntryDate = new Date().getTime();
                }
            }
            else if ( LineIn.indexOf("From:") != -1 ){
                NG.Author = LineIn.substring(
LineIn.indexOf("From:")+6, LineIn.length() );
                NG.Author = NG.Author.replace( ' ', ' ' );
            }
            else if ( LineIn.indexOf("Lines:") != -1 ){
                try{

```

```

        int
        Lines=Integer.parseInt(LineIn.substring(LineIn.indexOf("Lines:")+7, }
        LineIn.length()));
        if ( Lines > 120 )
            bBad = true;
        }catch(Exception E){}
    }

    //- Get the Body
    sendMessage( "BODY " + ID );
    NG.vBody = new Vector(30, 5);
    LineIn = In.readLine();
    if ( LineIn == null || LineIn.indexOf("222") == -1 ) return
    null;

    while ( (LineIn=In.readLine()) != null ){
        if ( LineIn.length() != 0 && LineIn.charAt(0) == '.' &&
        LineIn.length() == 1 )
            break;
        else if ( LineIn.length() != 0 && LineIn.indexOf("Content-
        type: ") != -1 ){
            if ( LineIn.indexOf("text/plain") != -1 )
                NG.vBody.addElement( LineIn + "\n" );
            else
                bBad = true;
        }
        else if ( LineIn.length() != 0 && LineIn.indexOf("begin
        600") != -1 )
            bBad = true;
        else if ( bBad == false )
            NG.vBody.addElement( LineIn + "\n" );
        }

    if ( bBad ) return null;
    NG.Newsgroup = Location + Type + cArticle.USENET;

    return NG;

    public void postArticle( String _NG, String _from, String _Sub-
    ject, Vector _Body )

    throws IOException {
        String LineIn;

        //- Get the Header
        sendMessage( "POST" );
        LineIn = In.readLine();
        if ( LineIn == null || LineIn.indexOf("340") == -1 ) return;

        //- Post the Header fields
        sendMessage( "Path: n-ary" );
        sendMessage( "From: " + _from );
        sendMessage( "Newsgroups: " + _NG );
        sendMessage( "Subject: " + _Subject );
        sendMessage( "Message-ID: <" + System.currentTimeMillis() +
        "@n-ary.com>" );
        sendMessage( "Date: " + new SimpleDateFormat( "dd MMM yy
        HH:mm:ss").format(new Date()) + " GMT" );
        sendMessage( "Organization: N-ARY Limited" );
        sendMessage( "Reply-To: " + _from + "\r\n" );

        //- Post the Body
        Enumeration E = _Body.elements();
        while ( E.hasMoreElements() )
            sendMessage( (String)E.nextElement() );

        sendMessage( "\r\n.\r\n" );
        In.readLine();
    }
}

```

Net
Guru
1/4 Ad

Object
Matter
1/4 Ad

DD SPR

31

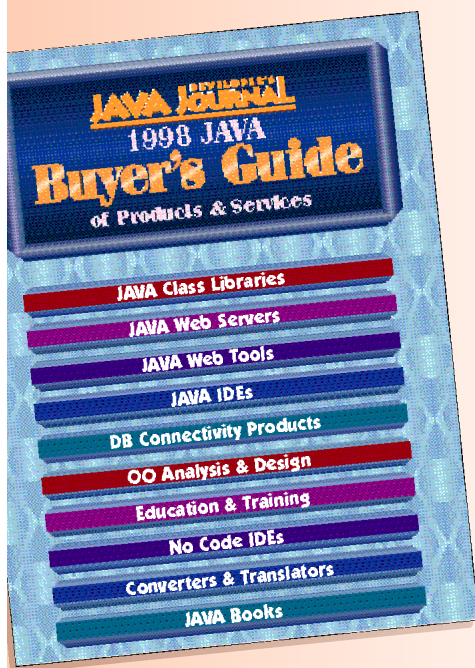
HEAD

ere's a Special
review of what
is to come!

sampling of the
1998 Java Buyer's
Guide to Products
& Services

rought to you
courtesy of

DEVELOPER'S
JAVA JOURNAL



**Coming
Soon!**

1998 JAVA Buyer's Guide to Products & Services

JAVA CLASS LIBRARIES

JSpell

JSpell is a 100% Java™ spell checking solution. It can be integrated with your applets and applications and is redistributable royalty free in your products. It includes two versions, standalone and client/server, which you can use depending on your applets and applications working environment.

Wall Street Wise Software

212 348-5031

www.wallstreetwise.com

Jelp

Jelp is a context-sensitive help system for Java™ applications/applets and is written entirely in 100% Java. Jelp converts RTF files, generated from your favorite help authoring tool, to Jelp booklets that may be distributed with our viewer in your application/applet royalty free.

CreativeSoft

Joe Spinelli

214 373-8720 FAX: 972 867-0111

www.jelp.com

Bitmap Graphics Pack

Display bitmap (BMP) images in your Java™ applets and applications. Bitmap Graphics Pack renders bitmaps in either original or specified size, and draws in the background which frees your app to do other tasks. Java 1.0.2 and Java 1.1 AWT compliant.

Solid Logic Computer Solutions, Inc.

Bob Barker

612 949-0140 FAX: 612 949-8602

www.slogic.com

DEVELOPMENT TOOLS

SQLSurfer

SQLSurfer is a development and deployment environment for corporate intranet/Internet Applications. It provides access to the components of corporate information systems and, in particular, relational databases.

Net@Way®

Regis Baudu

33 1 55 46 95 20 FAX: 33 1 55 46 95 29

www.sqlsurfer.com

J'Express

J'Express gives you Java™ installation and Web distribution. Click a button to create your cross-platform Java install program. If you like, use Java to customize your installation with special wizard panels or external programs.

DeNova, Inc.

Nan Atwell

408 490-2852 FAX: 408 490-2852

www.denova.com

Scriptic

Scriptic is a Java™ extension offering better support for parallelism and threading. It is aimed at user interfaces, simulations and research into parallelism. Scriptic comes with an extended Java™ compiler and a run-time library.

Delftware Technology BY

Andre van Delft

31 15 2578275 FAX: 31 15 2578387

www.delftware.nl

JAVA TESTING TOOLS

ProtoSpeed

ProtoSpeed's Distributed Java™ debugger provides the ability to examine distributed applications across the Internet, allowing developers to avoid time consuming 'trial and error' debugging. Use ProtoSpeed's Internet Protocol debugger to sit "on-the-wire" between a client and server to help prevent Internet protocol-related problems.

Progress Software Corporation

Adam Schwartz

781 280-4322 FAX: 781 280-4025

www.progress.com

TCAT for Java™

TCAT for Java™, a component tool of STW/Web, is a test coverage analysis tool configured for Java™ applets and for use on Java-enabled browsers. Developers of animated Web sites can use TCAT for Java to determine that their Java applets are fully exercised by their test suites – a critical quality verification when Java applets support financial transactions on the Web.

Software Research, Inc.

Jean Francis

415 957-1441 FAX: 415 957-0730

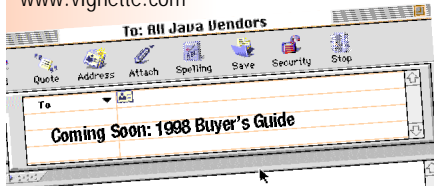
<http://www.soft.com>

WEB TOOLS

StoryServer 3

Vignette StoryServer 3 is a Web content application system designed to meet the demanding requirements of content-driven service applications. StoryServer 3 delivers a collaborative content management environment tightly integrated with a dynamic content application server.

Vignette Corporation
Jessica McMahon
512 502-0223 FAX: 512 502-0280
www.vignette.com



Each year Java Developer's Journal publishes our annual Java Buyer's Guide, the most comprehensive sourcebook and database of companies providing Java-related products and services available.

Throughout the year, our Buyer's Guide research staff exhaustively seeks out every company and person involved in providing goods and services to the Java development community. They then add it to our huge database, updating previous listings and removing listings that are no longer available. We then check our listings against those submitted through our interactive Company Product & Service Listing Update Service and publish the results annually. The Java Buyer's Guide is made available on-line in an efficient, complete and easily-searchable format that makes finding the most appropriate product or service easy for the user.

The Java Buyer's Guide is published by the same people who bring you Java Developer's Journal, the premier, independent, vendor-neutral monthly publication serving the needs of the entire Java development community. This is your assurance of the quality of the Java Buyer's Guide.

Jshrink

Jshrink removes names and unused data from compiled Java™ class files, resulting in smaller files that load faster and that yield less information when decompiled.

Jshrink does not change public or protected member names and can therefore be used with redistributable components, such as class libraries.

Eastridge Technology
Nick Eastridge
609 252-0825 FAX: 609 252-0826
www.e-t.com

JAVA IDES

WingEditor

WingEditor is a simple Java™ IDE implemented in pure-Java. It provides a Java-oriented editor environment, an error-browser capable environment and a GUI debugger.

WingEditor also supports NetRexx, a human-oriented language which is developed by IBM. WingSoft Company
Kathy Ho
510 744-1866 FAX: 510 494-0273
www.wingsoft.com

Visaj

Visaj is a new breed of Java™ development tool, written entirely in Java. Visaj gives developers the ability to build truly cross-platform Java applications, a key accomplishment in the platform-neutral world of the Internet and World Wide Web.

Imperial Software Technology
650 688-0200 FAX: 650 688-1054
www.ist.co.uk

Grinder: for Java™

Grinder for Java™ maximizes your programming power, software quality and portability, and object discovery and reuse. No other package allows you to explore Java from Sun, Microsoft or third party providers like Grinder.

The Paradigm Exchange
James Earle
770 395-1705 FAX: 770 395-1654
www.tpex.com

KAWA

KAWA is a simple yet powerful IDE for your Java™ development. KAWA is a wrapper on Sun's JDK. It is a Win 32 platform and is version independent. KAWA features include integrated syntax coloring editor, class browser, project manager, integrated context sensitive help and much more.

Tek-Tools, Inc.
Cindy Schlette
972 980-2890 FAX: 972 858-5348
www.tek-tools.com/kawa/

JAVA EDUCATION & TRAINING

Developer Kitchen

Phoenix Technologies' Developer Kitchens are technical, educational events that present Java™ developers with the landscape of tools and solutions available to them to enhance their development efforts.

Phoenix Technologies
Kimberly Morello
732 253-9000 FAX: 732 263-0189
www.phoenixtech.com

MindQ Developer Training for Java™

MindQ's Developer Training for Java™ is an effective developer-specific desktop training solution.

The product is a suite of network-hosted training courses, comprised of 15 modules with over 150 hours of training. MindQ Publishing, Inc.
Roger Bowman
408 927-4800 FAX: 408 927-4811
www.mindq.com

submit your
product for
inclusion in the
**1998 JAVA
BUYER'S
GUIDE**



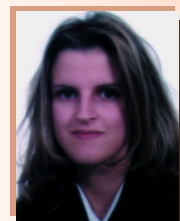
at
JavaBuyersGuide.com

Each year Java Developer's Journal publishes our annual Java Buyer's Guide to Products & Services, the most comprehensive sourcebook and database of companies providing Java-related products and services available. Over 200,000 purchasers of Java products and services will see your company's listing in the 1998 Java Buyer's Guide.

To have your company's product or service included in the 1998 Java Buyer's Guide to Products & Services, simply fill out our online listing submission form.

Point your Web browser to www.javabuyersguide.com to have your product or service listed today!

Contact



Christy Wrightington
at 914-735-7300
or Christyw@sys-con.com.

Multi-Threading in Java

Programming that requires a different way of thinking – but offers great rewards

by Tod Cunningham

Introduction

Multi-tasking is rapidly becoming a necessity in software development today. All major operating systems support some form of multi-tasking, and as costs come down it is becoming common for high end systems to incorporate multiple processors.

Multi-Tasking and Threads

At its most basic level, multi-tasking allows multiple programs to be run at the "same" time. The best way to visualize this is to think of each application as running on its own processor.

It would be quite inefficient for each application to have a dedicated processor. A major function of most modern operating systems is to make each application share access to processors by preempting one application to let another one run. Figure 1 illustrates the difference between processor sharing and non-sharing.

Just like programs can run concurrently, pieces of the same program can run concurrently. This ability is known as threading and it is what Java supports. Figure 2 illustrates how a program can be threaded.

Threads are becoming more popular because they are faster to set up, often require less memory and allow better encapsulation.

Commonly, it is the responsibility of the operating system to schedule and preempt each thread, just like it preempts each application. This usually leads to platform-specific methods of multi-threading.

Platform-Independent Threading

Most programming languages rely on operating system-specific calls to support multi-threading. For example: C/C++ programs in Unix often use `fork()` and Window

95/NT C/C++ programs often use `CreateThread()`. This can cause a lot of headaches when trying to port an application.

Since one of Java's goals is to "write once, run anywhere", the Java language specification contains support for threading. In theory, this allows multi-threaded programs to be run on any platform which supports Java without concern for how the Java Virtual Machine (JVM) actually implements the threading.

Most JVMs don't actually use native operating system threads to implement threading. They implement their own task scheduling and context switching algorithms within the JVM. This makes the JVM easier to port from one operating system to another. However, Sun is going to be releasing a JVM for the Sun Solaris SPARC which uses native threads to implement Java threads. Having the JVM use native threads can be a real benefit to Java applications because of gained responsiveness to other running processes. Best of all, a Java application doesn't have to do anything special to make use of benefits supplied by different implementations.

Introduction to Java Threading

Java is one of the few common-programming languages that actually supports threading in the language itself.

Java defines a `Thread` class and a `Runnable` interface that can be used to define a thread object. Take a look at the Basic Thread example that derives from the `Thread` class. It creates two threads that display messages asynchronously.

Run this example multiple times and under different JVMs and see how and when the messages are displayed.

Depending on the speed of your machine and the JVM you are using, it may appear that the threads are not being preempted. For example: All of the first thread's messages may be printed followed by all of the second thread's messages.

This usually happens when running on a fast computer or using Just-In-Time (JIT) Java. The reason for this is that one of the sample threads may actually finishing printing all of its messages before it is scheduled for preemption. Try making the threads take longer to finish by increasing `MAX_INDEX` to 100 (or more) and see what happens.

Runnable Interface

The Basic Thread example derives from the `Thread` class to create a thread object; However, a class may implement the `Runnable` interface instead.

Implementing the `Runnable` interface is useful when a class needs to be multi-threaded and also be derived from another class. Remember that Java supports only single inheritance.

To change the Basic Thread example to use a `Runnable` interface, change the `CountThread` definition to:

```
public class CountThread implements Runnable
```

Then, change the `CountThread` object declarations to:

```
CountThread countRunnable1 = new Count-
Thread( "Thread 1" );
CountThread countRunnable2 = new Count-
Thread( "Thread 2" );
Thread countThread1      = new Thread(
CountRunnable1 );
Thread countThread2      = new Thread(
countRunnable2 );
```

This works because the `Thread` class supports a special constructor that accepts a `Runnable` interface. Therefore, we can create a thread based on a `Runnable` interface. The only method that

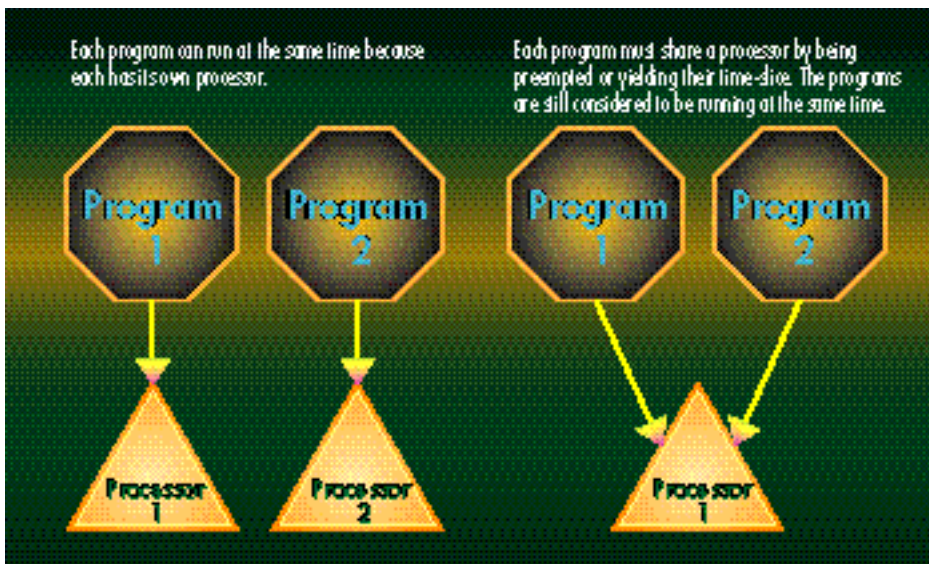


Figure 1: Difference between processor sharing and non-sharing

the interface defines is run().

start() and run()

By calling a Thread object's start() method a Java application tells the JVM to start a separate thread of execution. The JVM will only allow a Thread object to create a single thread of execution for the lifetime of the object. Subsequent calls to an object's start() method will be ignored if the thread associated with the object has terminated; otherwise, the JVM will cause the start() method to throw IllegalStateException.

Once the JVM sets up the separate thread of execution it will call the object's run() method from within the newly created thread. The run() method of a Java Thread is like the thread's "main" method. Once started by the JVM, the thread exists until the run() method terminates.

stop()

The stop() method is used to stop the execution of a thread before its run() method terminates.

However, the use of stop() is discouraged because it will not always stop a thread. The stop() method is a synchronized method and as such will not stop other synchronized method blocks. This means that a deadlocked thread can't be stopped, which isn't very useful. Synchronized methods are discussed in the section on monitors.

The best way to stop a thread is to let the run() method exit gracefully by using proper synchronization techniques like the ones that follow.

join()

Join is a simple synchronization mechanism that allows one thread to wait for another to finish. In the Basic Thread exam-

ple, the main application waits for the threads that it started to finish. Note that the order in which threads are joined is not important.

Thread Synchronization

Writing multi-threaded applications usually involves much more than just starting and stopping threads. Usually some form of thread synchronization is required at key points in time. There are two main types of synchronization that Java supports: Monitors and Mutexes.

Monitors

The term monitor comes from the monolithic monitor (more commonly known

access by other threads).

Each Java object has a monitor and only one thread at a time has access to that monitor. When more than one thread wants access to an object's monitor, they must wait until that monitor is released. Notice that the Thread object itself has nothing to do with implementing monitors. Monitors are inherent in every Java object, and every Java object has its own independent monitor.

Java defines the keyword "synchronized" to gain access to an object's monitor. There are two ways to use synchronize: either by method or by block.

To allow only one thread at a time to access an object's method, use the "synchronized" keyword in the definition of the method.

```
public synchronized void sem_wait( String
currentThreadName )
{
    // Statements here are under protection of
    // the object's monitor.
    // Each instance of the object will allow
    // only one thread at a time access to the
    // method.
}
```

To allow only one thread at a time access to a portion of an object's method, use the block form of synchronized within the method (see Listing 5).

Notice that the block form of "synchronized" takes an object as a parameter. The monitor associated with the given object is

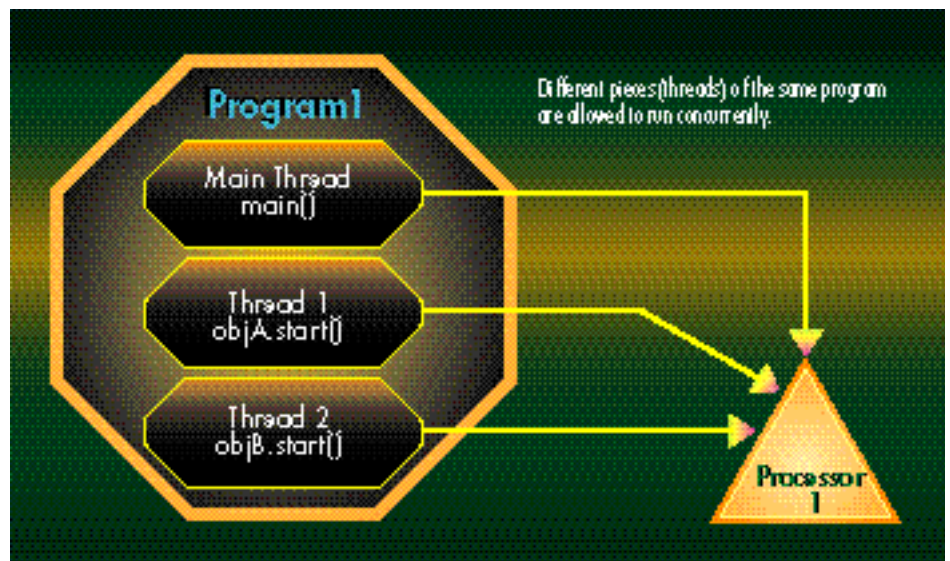


Figure 2: How a program can be threaded

today as a kernel) found in operating systems. A fundamental responsibility of an operating system is to protect system resources from unrestricted access, much like a monitor protects the internal data and methods of an object from unrestricted

used to perform the synchronization.

Although we can specify individual methods and blocks to be synchronized, there is still only a single monitor per object. Once a thread enters a synchronized section of an object it has acquired

that object's monitor. Since the object's monitor is now acquired, all other threads trying to acquire that monitor will have to wait until it is released. The monitor will be released when the one thread that entered the object's synchronized section leaves the section.

While a thread has acquired an object's monitor, it will immediately succeed in subsequent attempts to acquire that same object's monitor. This makes sense because the purpose of monitors is to allow only a single thread access to some section of code. Since the thread already has access to the monitor it is safe to let that thread execute the code. This is very useful because it means that an object's synchronized methods may call each other without fear of delay or deadlock-ing.

If for some reason a thread acquires an object's monitor and doesn't release it, the waiting threads will wait "forever". This is called a deadlock and can occur very easily. Deadlocks can be hard to find in code and may not show up under testing depending on the timing of the threads. It is recommended that a monitor be used to protect only what is absolutely necessary for correct behavior. Use the synchronized block mechanism to limit the scope of the monitor. In addition, don't call any methods

within a synchronized block except the class Object methods (wait, notify,). This will greatly reduce the chances of deadlocks.

Remember that monitors are based on objects so be careful when using references to objects. Each reference to an object uses the monitor of the object being referenced.

One complexity of monitors is that static methods may also be synchronized. However, a static method is not associated with an object. To handle this, all static synchronized methods of a class share a single monitor that works independently of an object's monitor. When a synchronized method calls a static synchronized method, it must acquire another monitor (the one associated with all static synchronized methods of the class).

Mutexes

Mutexes are used when two or more threads can't interleave certain types of operations. Thus, one sequence must be completed before the other is started. The join() method, used in the Basic Thread example, was a simple fixed use mutex that waits for the thread being joined to stop.

The generic mutex methods: wait(), notify() and notifyAll() are available to all Java objects because they are declared in the Object Java class. These methods allow any

thread to wait for any other thread to complete some activity. When the activity is complete, the thread notifies one (or all if notifyAll is called) waiting threads.

wait()

The thread that calls an object's wait() method will be suspended and any monitors the thread had acquired will be released. The thread will remain suspended until it is notified and the monitors needed by the thread can be reacquired.

In order for a thread to call an object's wait() method, it must own the object's monitor.

notify() and notifyAll()

A thread calls an object's notify() method when it wants to let a thread waiting on that object know that some activity has been completed. A waiting thread will be awakened and put back in the queue of running threads. However, the awakened thread still has to reacquire all monitors that it released when it called wait().

The thread that calls an object's notify() or notifyAll() method must have possession of the object's monitor.

When notifyAll() is called, all threads that are currently waiting on the object's monitor will be awakened.

Listing 1: Basic thread.java

```
import java.lang.*;
import java.io.*;

/*
** Starts two threads which print messages asynchronously.
*/
public class BasicThread
{
    public static void main( String args[] ) throws
    IOException, InterruptedException
    {
        CountThread countThread1 = new CountThread( "Thread 1" );
        CountThread countThread2 = new CountThread( "Thread 2" );

        System.out.println( "Starting Threads" );
        countThread1.start(); // Starts thread and calls run
        countThread2.start(); // Starts thread and calls run

        System.out.println( "Threads Started" );

        countThread1.join(); // Wait for thread to end.
        countThread2.join(); // Wait for thread to end.
        System.out.println( "Threads Stopped" );

        System.out.println( "Press <enter> to quit." );
        System.in.read(); // Wait for user to press enter
    }
}
```

Listing 2: countthread.java

```
import java.lang.*;
```

```
/*
** This class prints messages to the system console. Objects
** of this class may be run as separate threads by calling
** the thread method start().
*/
public class CountThread extends Thread
{
    final int MAX_INDEX = 10;
    protected String currentThreadName;

    public CountThread( String threadName )
    {
        currentThreadName = threadName;
    }

    public void run()
    {
        int index;

        for( index = 0; index < MAX_INDEX; index += 1 )
        {
            System.out.println( currentThreadName + ": " + index );
        }
    }
}
```

Listing 3.

```
public void sem_wait( String currentThreadName )
{
    // Statements here are not run under protection of a monitor
    synchronized( this )
    {
```


Semaphores

One of the problems with Java Mutexes is that notify will only wake up threads that are currently waiting. This can cause synchronization headaches because one must make sure that a thread waits before notify is called. In other words, the notification is lost when there is no one waiting. While not directly supported by Java, a semaphore can be emulated to solve this problem.

Just like mutexes, semaphores are used when two or more threads can't interleave certain types of operations. Thus, one sequence must be completed before the other is started. However, a semaphore contains more state information that allows it to overcome the limitations of Java mutexes.

We can emulate the most common form of semaphore, called a blocked-set semaphore, by using monitors and mutexes. The blocked-set semaphore has the following definition shown in Listing 4. A single thread awakens one suspended thread.

See the example Semaphore.java for an implementation of this semaphore.

Notice that we will only call notify() if we have first done a wait(), and we will remember when we called sem_signal() without wait() being called.

Putting it all together

One of the classic concurrent programming problems is the producer/consumer problem. It involves two threads: one producer thread and one consumer thread. Take a look at the producer/consumer example.

The producer produces integer numbers and prints a message that states an integer was produced. The consumer consumes an integer number, supplied by the producer, and prints a message that states an integer was consumed. When the consumer receives the product (number) 0 it knows the producer is done and quits.

A small integer item buffer is used so that the producer can make multiple products without having to wait for the consumer to consume them.

The example just produces integer numbers to keep the example concise. However, the producer/consumer principle can be used to solve many real world problems. A producer could search for files and produce found filenames to a consumer window. A producer could do database queries that send results to a report window. There are countless concurrency problems that may be solved with this technique.

Conclusion

Java developers not only get a great object-oriented language, but also get a language that supports multi-threading. However, just like good object-oriented development requires a different way of thinking, good threaded programming requires a different way of thinking – with the rewards just as great.

Be creative and remember that all aspects of Java can be threaded to solve everyday problems: Windowing interfaces (AWT), saving and loading files (File I/O) and reusable components (Beans™). 🍌

References

M. Ben-Ari, "Principles of Concurrent and Distributed Programming", Prentice Hall, New York, 1990.

S. Oaks & H. Wong, "Java Threads", O'Reilly, MA 1997

About the Author

Tod Cunningham is a software engineer with development experience in real-time multi-processing systems written in C/C++. as well as Java. He graduated from the University of Toledo in 1992 with a BS in Computer Science and Engineering.



tcunning@earthlink.net

```
// Statements here are under protection of the object's monitor.
// Each instance of the object will only allow one thread at a
time access to the method.
}
// Statements here are not run under protection of a monitor
}
```

Listing 4: Definition of blocked-set semaphore.

```
obj.sem_wait():if( obj.s > 0 )
    obj.s = obj.s - 1;
else
    obj.wait(); // Suspend

obj.sem_signal():if( thread suspended on this semaphore )
    notify(); // Awaken waiting thread
else
    obj.s = obj.s + 1;
```

Listing 5: Semaphore.java

```
/*
** The class implements a blocked-set semaphore which can be used
** for thread synchronization.
**
** A blocked-set semaphore has the following definition:
**
** A single thread awakens one suspended thread.
**
** obj.sem_wait():      if( obj.s > 0 )
**                      obj.s = obj.s - 1;
**                      else
**                      obj.wait(); // Suspend
**
** obj.sem_signal():    if( thread suspended on this sema-
```

```
phore )
**                      notify(); // Awaken waiting
thread
**                      else
**                      obj.s = obj.s + 1;
* /
public class Semaphore
{
    private int    s;
    private int    numWait;
    private String currentSemaphoreName;
    private boolean doDebug;

    public Semaphore()
    {
        s = 0;
        numWait = 0;
        currentSemaphoreName = this.toString();
        doDebug = false;
    }

    public Semaphore( String name, int initial, boolean debug )
    {
        s = initial;
        numWait = 0;
        currentSemaphoreName = name;
        doDebug = debug;
    }

    public void sem_wait()
    {
        sem_wait( Thread.currentThread().getName() );
    }
}
```

```

    }

    public synchronized void sem_wait( String currentThreadName )
    {
        boolean retry;

        / *
        ** Check to see if semaphore was signled while no thread
was waiting.
        * /
        if( s > 0 )
        {
            s = s - 1;    // Match up one signal with one wait
            return;
        }

        numWait = numWait + 1;    // Mark the we are going to
wait.
        do
        {
            retry = false;
            try
            {
                if( doDebug )
                    System.out.println( currentThreadName + " Waiting
for " + currentSemaphoreName );

                this.wait();    // Do a Java wait() mutex.

                if( doDebug )
                    System.out.println( currentThreadName + " Received
" + currentSemaphoreName );

            } catch( InterruptedException e ) { retry = true; }
            } while( retry );
        }

        public void sem_signal()
        {
            sem_signal( Thread.currentThread().getName() );
        }

        public synchronized void sem_signal( String currentThreadName )
        {
            boolean retry;

            / *
            ** Check to see if a thread is waiting for a signal.
            ** If not, then remember that a signal was generated with-
out
            ** a wait.
            * /
            if( numWait == 0 )
            {
                s = s + 1;
                return;
            }

            numWait = numWait - 1;    // We are going to wake up one
waiting thread.
            do
            {
                retry = false;
                try
                {
                    if( doDebug )
                        System.out.println( currentThreadName + " Notify

```

```

for " + currentSemaphoreName );

                this.notify();
            }
            catch( Exception e ) { retry = true; }
        } while( retry );
    }
}

```

Listing 6: Producer.java

```

import java.lang.*;
import java.math.*;

public class Producer extends Thread
{
    final int            MAX_PRODUCT_BUFF = 4;
    volatile private int product[] = new int[MAX_PRODUCT_BUFF];
    volatile int         count;
    int                  localCountIn;
    int                  localCountOut;
    int                  inIndex;
    int                  outIndex;
    volatile Semaphore   notFull = new Semaphore( "notFull", 0,
true );
    volatile Semaphore   notEmpty = new Semaphore( "notEmpty", 0,
true );

    public Producer( String threadName )
    {
        setName( threadName );
        count = 0;
        inIndex = 0;
        outIndex = 0;
        localCountIn = 0;
        localCountOut = 0;
    }

    public void run()
    {
        final int MAX_PRODUCTION = 10;
        int index;

        for( index = 0; index < MAX_PRODUCTION; index += 1 )
        {
            make_product( index + 100 );
            System.out.println( getName() + ": " + (index + 100) );
        }
        make_product( 0 );    // Signal end of production
        System.out.println( getName() + ": done" );
    }

    public void make_product( int value )
    {
        if( localCountIn == MAX_PRODUCT_BUFF )
            notFull.sem_wait();

        product[inIndex] = value;
        synchronized( this )
        {
            count = count + 1;
            localCountIn = count;
        }

        if( localCountIn == 1 )
            notEmpty.sem_signal();
    }
}

```

```

inIndex = (inIndex + 1);
if( inIndex == MAX_PRODUCT_BUFF )
    inIndex = 0;
}

public int take_product()
{
    int value;

    if( localCountOut == 0 )
        notEmpty.sem_wait( currentThread().getName() );

    value = product[outIndex];
    synchronized( this )
    {
        count = count - 1;
        localCountOut = count;
    }

    if( localCountOut == MAX_PRODUCT_BUFF - 1 )
        notFull.sem_signal( currentThread().getName() );

    outIndex = outIndex + 1;
    if( outIndex == MAX_PRODUCT_BUFF )
        outIndex = 0;

    return( value );
}
}

```

Listing 7: consumer.java

```

import java.lang.*;

public class Consumer extends Thread
{
    private Producer theProducer;

    public Consumer( String threadName, Producer producer )
    {
        setName( threadName );
        theProducer = producer;
    }

    public void run()
    {
        int index;
        int value;

        do
        {
            value = theProducer.take_product();
            if( value != 0 )
                System.out.println( getName() + ": " + value );
        } while( value != 0 );
        System.out.println( getName() + ": done" );
    }
}

```

Net Guru 1/2 Ad

How: Spr

se Ad

ead

Ultrrex Announces CruXpert, Java-Based Expert System Development Tools in a Box

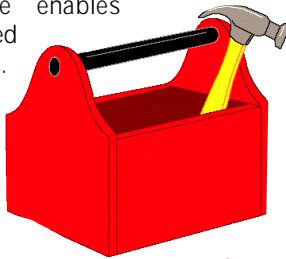
(Seattle, WA) - Ultrrex Corp. has released CruXpert™ Version 2.0, a software product for the development of knowledge-based applications deployable over the Internet or corporate Intranet.

Targeted at Web site designers and Web masters, this software enables deployment of "automated experts" as Java applets. This will help the user to quickly find the solution to their problem interactively.

CruXpert 2.0 is the only development tool on the market that provides Rapid Application Development (RAD) or expert system applications in Java. It facilitates the specification of rules into a knowledge base in an easy-to-learn English-like language which can be used by a lay person with no previous programming experience. CruXpert compiles the knowledge base written in the language into an expert system application in Java. Typical applications are troubleshooting and diagnostics, configuration, sales support and advisory.

CruXpert requires JDK 1.0.2 and applets created with this software can be viewed with any Java-compatible browser such as Netscape™ 3.0+ or Internet Explorer™ 3.0+. Four demonstration applets are included for easy modification to developer needs. A free version is available for trial. Complete products are available as "Lite," "Standard" and "Pro" editions at \$99, \$299 and \$995 respectively.

For more information, visit the Ultrrex Web site at www.ultrrex.com.



time off to sit in a classroom and travel.

The custom-designed courseware is based on MindQ's "Complete Java Training Series," a suite of interactive, CD-ROM-based training titles designed to help corporations quickly train applications developers and software engineers.

Novell is the world's leading provider of network software. For more information on their products and services, visit their Web site at www.novell.com. MindQ Publishing Inc develops and markets interactive training products for developers. For more information on their products, call 800 MIND-008 or visit their Web site at www.mindq.com.

GraphOn, Prologue Software Debut GO-Between

(Las Vegas, NV) - GraphOn and Prologue Software have developed "GO-Between" in response to Microsoft's interest in interoperability and migrating UNIX enterprises to Hydra. GO-Between allows enterprises to preserve their investment in UNIX applications while taking advantage of the features of multi-user NT. It is part of GraphOn's ongoing efforts to bring diverse multi-user operating systems together. Distribution and pricing for GO-Between have not been announced yet.

GraphOn has also signed a letter of intent to merge with Prologue Software, a French multi-user software company. Prologue Software will soon deploy the GO-Between technology to its customers.

For more information on GraphOn, see their Web site at www.graphon.com. For more information on Prologue Software, see their Web site at www.prologue-software.fr.

Superscape Endorses the Java 3D API

(Santa Clara, CA) - As part of its continued commitment to support existing and emerging standards, 3D tool vendor Superscape, Inc. has endorsed the use of the Java 3D API in the development of 3D content for Web delivery.

Superscape has more than ten years of experience in designing and developing real-time interactive 3D authoring software.

Their recent move towards supporting the VRML standard has met with approval from both the industry and their clients, who welcome additional flexibility and choice in their selection of authoring environment.

The Java 3D API, released by Sun, provides a set of object-oriented interfaces for Java™ applications that require high performance, cross-platform, interactive 3D graphics. It is designed to fulfill the growing demand for full-featured multimedia capabilities which can be seamlessly integrated into the browser environment.

Superscape develops and publishes interactive 3D authoring software for the World Wide Web and standalone 3D applications. They also publish the Web browser Viscap and provide 3D content creation services.

For more information on Superscape, visit their Web site at www.superscape.com and the Virtual World Wide Web, www.vwww.com.

AccordNet's Hyperbanner™ Provides Global Internet Advertising Solutions

(New York, NY) - AccordNet Ltd. has introduced its advanced JAnimation™

(Java-Animation) advertising banners. The HyperBanner Network is the largest international (non-US) Internet banner advertising network.

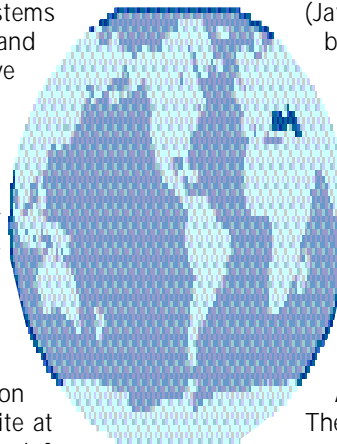
AccordNet has developed a sophisticated database-driven banner system that allows advertisers to focus on specific geographical regions or areas of interest, such as Software, Music, Sports and Martial Arts.

The JAnimation banner technology enables dynamic, animated banners that typically "weigh" only 1-3K for 20-30 seconds of animation.

For more information on JAnimation, see their Web site at www.hyperbanner.net.

Borland to Acquire Visigenic To Speed Growth in Enterprise Computing Market

(Las Vegas, NV) - Borland International, Inc., a leading supplier of application development technologies for client/server and multi-tier applications, has signed a definitive agreement to acquire Visigenic Software, Inc., the leading provider of Object



Request Broker (ORB) technology to the computer industry, in a stock-for-stock acquisition.

Borland's proposed acquisition of Visigenic is intended to establish the company as a leader in distributed enterprise computing. A distributed computing architecture is necessary to tie disparate enterprise systems together. Middleware is the key technology used to connect the business logic and components on the middle tier with client-based user interfaces and database management systems. Visigenic is a leading supplier of object middleware based on the CORBA standard. They also offer a leading implementation of VisiBroker for C++ with complete interoperability.

By combining Borland's object-oriented, development environments with Visigenic's distributed object middleware, the company will be able to offer corporate IT a complete, integrated solution for developing, deploying and managing distributed enterprise applications.

For more information on Borland, see their Web site at www.borland.com. For more information on Visigenic, see their Web site at www.visigenic.com.

Live Software Releases JRun 2.0 for IIS and Netscape Servers

(San Diego, CA) - Live Software Inc. has released version 2.0 of its JRun product line for Microsoft's Internet Information Server and Netscape's Enterprise and FastTrack 3.0 servers for x86.

JRun is a combination of native code and Java code that adds support for the Java Servlet API to these Web servers. This version offers the option of using either the Sun Java Virtual Machine or the Microsoft Virtual Machine for Java on either server. It is available for all platforms for commercial use at no cost.

By supporting a wide spectrum of Web servers, JRun assures the developer of complete cross-platform and cross-server compatibility, allowing development on one server/platform and deployment on another. All versions of JRun 2.0 fully implement the Java Servlet API.

JRun 2.0 for Windows 95/NT may be downloaded from the Live Software Web site at www.livesoftware.com. The version for the Microsoft Virtual Machine for Java and Netscape FastTrack and Enterprise servers for Solaris, HP-UX, IRIX and AIX is also available.

For more information on Live Software, e-mail info@livesoftware.com or see their Web site at www.livesoftware.com.

FuegoENGINE™ and Fuego-BUILDER™ Provide Process Management and Component Development Solutions

(Dallas, TX) - Fuego Technology Corp. has introduced two new 100% Pure Java™ products. FuegoENGINE, its flagship products, is the first enterprise management system that uses Java™ technology to automate mission-critical business processes and coordinate and integrate disparate software applications. FuegoBUILDER is a business process management system that provides a robust, open platform for deploying new JavaBeans™ components alongside legacy, client/server and office applications.

FuegoBUILDER is a JavaBeans component-based development environment that offers several advanced features for customers that need to deploy application components in limited bandwidth settings. Together with FuegoENGINE, it will offer an extremely powerful platform-independent solution available for applications integration through enterprise process management. Both products have 100% Pure Java certification. Commercial availability is planned for early 1998.

For more information, see the Fuego Technology Corp. Web site at www.fuegotech.com.

Mainstay Announces PageMover™ - Java Alternative to DHTML

(New York, NY) - Mainstay has announced PageMover™, a family of customizable Java applets designed to offer a cross-browser, cross-platform, extensible alternative to Dynamic HTML (DHTML). PageMover Java applets offer DHTML-like control objects and interactive display "billboards" that require no programming or scripting. They work in both Microsoft Internet Explorer and Netscape Navigator equally and without modification. They are also WYSIWYG customizable in real time through Mainstay's included AppletSet'r technology - directly within the Web browser.

PageMover gives Web designers the ability to add DHTML-like controls and interac-

tive "billboards" without the need to wait for a realized DHTML standard. It requires no specialized plug-ins or third-party applications, is compact, less than 10KB on the average, and optimized for fast, responsive Web pages. The applets are designed to offer maximum variability.

PageMover will be available from Mainstay in Q1 1998, with a street price of approximately \$149. For more information, contact Lance Merker at 805 484-9400 or by e-mail at lance@mstay.com.

NetDynamics Introduces Network Application Platform

(Washington, DC) - NetDynamics, Inc. has announced NetDynamics 4.0, its next generation Enterprise Network Application Platform. It has breakthrough enterprise technology integration and real-time distributed application management coupled with the best solution for scalability and application delivery. Built from the ground up on industry standards CORBA and Java, this platform delivers the ability to leverage existing software and hardware investments into innovative network applications.

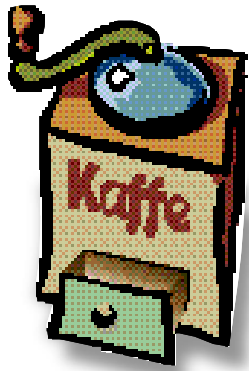
The NetDynamics Enterprise Network Application Platform is available in a starter package that includes five developers, a 25-user Application Server and WebAssist support for \$14,000. Components of the platform can also be purchased separately. For more information, visit their Web site at www.netdynamics.com.

Correction

The following are the authors of the "Anything Under the Sun" column which appeared in JDJ, Volume 2, Issue 11:

Achut Reddy is a Staff Engineer in the Authoring and Development Tools group, currently working on Java performance issues. The material in the article is part of a book tentatively titled "Fast JAVA: Performance Tuning Guide" to be published in mid-1998 by Sun Microsystems Press/Prentice Hall. He can be reached at achut.reddy@sun.com.

Daniel McKinley Lord is the Technology Manager for Java and Developer Products in SunSoft's Worldwide Systems Engineering Group. He holds a BSEE and an MBA. You may email him with questions and comments at daniel.lord@sun.com.



THE GRIND

by Joe S. Valley

"So, almost two
years after its
introduction, Java is
a great programming
language, and
nothing more."

Joe S. Valley is a scarred veteran of the Silicon Valley wars. It was either writing this column or heading back into therapy. His company can't afford mental health care coverage anymore, so writing is the only option. There are a million stories in the Valley and Joe knows lots of them. Got a good story? E-mail him at Joe@sys-con.com



Joe@sys-con.com

Prophet of Doom

Joe here again, moonlighting as the Man-on-the-Street reporter for Java Developer's Journal. Things are winding down at the day job. My staff is quitting so fast, they are leaving skid marks in the parking lot.

Me? I'm spending a lot of time at trade shows and those industry events that are of little value when you have real work to do, but are fun when you don't. Funny thing is that all of the events seem to feature the same two speakers professing Java and saying nasty things about Microsoft. I guess Larry Ellison of Oracle and Scott McNealy of Sun are trying to make it as stand-up comedians. I have heard all of their jokes about the twelve ways that Microsoft Word can underline a misspelling. Larry and Scott speak without notes, wander around the stage in circles and say the same things about Bill Gates and Java over and over. Reminds me of Rodney Dangerfield in his heyday. He told the same jokes every time you saw him on TV. The difference was that Rodney was funny. Larry and Scott? They are just boring.

By their actions, the Prophets of Java are sending a different message now:

Java is DEAD.

WHAT? DEAD? Well Java the LANGUAGE is alive and doing quite well, thank you. The programming tools continue to sell; books and magazines, such as Java Developer's Journal, are in a bull market. But Java the ENVIRONMENT is on life support. Virtually no one has made a profitable business with Java, the ENVIRONMENT. No profit, a doomed future.

Java, the LANGUAGE, has found a niche as Esperanto in the multi-standard client/server world. The ability to embed Java applets in a Web page has and will continue to create new opportunities for the browser vendors. The tool vendors, such as the vilified Microsoft, will also do well with Java products.

But in so far as the Network Computers (NCs), set-top boxes, PDAs or Mood Rings, Java is near death. Larry and Scott are wrong. Bill Gates and Windows are winning...again.

NCs are running server-based Windows apps, not Java applets. PDAs are running lean and mean OS's or Windows CE, not JavaOS. Heart pacemakers and Boeing 757s

are running ultra-reliable Real-Time Operating Systems (RTOS's) like pSOS. Users of the much-hyped JavaOS, pJava and eJava are nowhere to be found, except in the fuzzy world of press releases.

Can Java, the ENVIRONMENT be saved? Should it be saved? Here is Joe's humble advice to Sun on what needs to be done to extend the success of Java, the LANGUAGE, into new areas.

1. Forget about writing operating systems for Java. Let Microsoft, IBM and the RTOS vendors do that. JavaOS has gone into the Roach Motel of software, SunSoft, and will never come out. Let it die.
2. Continue to support the JavaVM on all OS's, but with cheaper licensing models and more reasonable support fees. This will stop the Java clones from fragmenting the market. RTOS vendors cannot economically justify the Java licensing fees and may be forced to go with Java clones if their customers need an embedded Java solution. Fragmented markets make confused developers. Confused developers will stay with Windows and not take risks with Java on any other OS's.
3. Get someone, anyone, to write a decent, simple, Java-based office suite. This is the area where Microsoft is most vulnerable. The new Microsoft Office 97 is horrible to install and use but it is the only game in town. Use MacWrite as the reference for the word processor. This suite will encourage the use of NCs with the JavaVM on whatever OS is appropriate. Microsoft may get the OS part of the sale, but Sun and others will get the rest of it.
4. Have a consistent Java message. Sun divisions are not in sync, especially JavaSoft and Sun Microelectronics (picoJava chips). If Sun isn't clear about Java, then who will be? Where are your marketing people hiding? This is Product Marketing 101 level stuff.

So, almost two years after its introduction, Java is a great programming language and nothing more. If Java is to move beyond the world of COBOL, Basic and APL, Sun needs to move fast. Moving fast is not what Sun has been able to do.

What about old Joe? I continue to go to these industry events, eat rubber chicken and mystery meat for lunch and ponder the fate of Java. ☘

JavaWorld Ad

KL Group Full Page Ad