

JAVA™ DEVELOPER'S JOURNAL

The World's Leading Java Resource

February 2001 Volume: 6 Issue: 2

JAVADEVELOPERSJOURNAL.COM



From the Editor
by Sean Rhody pg. 5

Macromedia Focus
by Kevin Lynch pg. 7

Industry Watch
by Alan Williamson pg. 70

VisualAge Repository
by Brady Flowers pg. 82

JDJ News
pg. 98

Product Review
EspressChart
by Don Walker pg. 94

Java Jobs
by Bill Baloglu &
Billy Palmieri pg. 104

Guest Editorial
by Ajit Sagar pg. 110

RETAILERS PLEASE DISPLAY
UNTIL APRIL 30, 2001
\$4.99US \$6.99CAN



Feature: A Practical Solution for the Deployment of JSP *Develop a solution that's portable* **PART 2** Alexis Grandemange **22**

EJB/CORBA: EJB, CORBA, and COM *Maintaining interoperability is essential* Sirl Davis **30**

EJB Home: Modeling Enterprise Java Components with UML *Finding common ground* Vaughn Vernon **38**

Java & HTTP: Journeyman's HTTP Driver *A portable and effective means for generating HTTP traffic* Marc Connolly **46**

CORBA Corner: Integrating CORBA and J2EE *Integration servers help fill the gap in the J2EE-CORBA combo* Paul Moxon **54**

Feature: Business Rule Representation of Java Objects *Representing data in automated systems* Ken Molay **62**

Feature: Jlink: Cybelink's Framework for Creating Reusable Enterprise Components *Exploring Servlet/JavaServer Pages technology* Mani Malarvannan **74**

Feature: Using Motorola's Java Card to Digitally Sign a Message *Integrating Java, smart cards, & cryptography* Andrew Webb **86**

SEAN RHODY, EDITOR-IN-CHIEF



Impersonalization

You see personalization and targeted marketing all over the Web. Almost every commerce site offers you the opportunity to set up your own favorites, rearrange their home page to suit your tastes, and be remembered when you come to their site. Every site I visit allows me to set up my own personalized content. I use MSN for some things, like tracking my stocks and local weather. I use CNN for news. I use Amazon for buying things and eBay for trading. And everyone lets me do it my way.

As a system architect who concentrates on commerce sites, I spend a lot of time figuring out how to do the very same things. Or more appropriately how to use existing products to do what the client wants to do.

What they really want is to get to know you better. They want to know who you are, your age group, your sex is, how much money you make, where you live, and your shoe size. Not all of them want all that information, of course, but you get the idea.

To gather this information, different sites try one of two approaches; both have limitations. The first approach is to ask the person for specific information. Unfortunately, people lie. Half of the programmers I've worked with in the industry have listed themselves as CIO at one time or another on a magazine form in order to get a free subscription (*Note: That won't work with JDD*). And how many people really check off the lowest income bracket in the section that asks how much money you make? Not many. When people aren't happy with providing such information, they either don't do it or they lie.

The other method that sites try is to implicitly derive information about you. Every time you do a search or buy an item, you may unintentionally be giving information about yourself. Unfortunately, this is far from foolproof too. I have a friend who always complains about a particular shopping site. He's a single guy, but he bought a children's book for a friend's daughter once and ever since he receives recommendations for children's books every time he visits the site. He's in their "has children" category.

There's nothing inherently wrong with sites using either approach. Commerce sites are in business to make money, and the more they get to know their customers, the better they can serve them. Unfortunately, the products that are available to help target their marketing efforts have a dark side – they require people.

As far as I know, no one has ever developed a computer system that can make a judgment call. Computers are great tools for evaluating conditions and generating results, but they can't tell that my friend doesn't have kids. The biggest mistake a commerce site can make is to think that a package can reduce the number of people they need in marketing.

It's obvious if you think about it for a minute. The more finely tuned you want your marketing and sales to be, the greater the number of categories. Computers can't create the categories any more than they can determine the conditions under which a customer belongs to one of them. A human being has to do it and input it into the system. People also have to create the business rules for the various special offers. They need to decide that a good cross-sell for children's books might be children's software, and whether a book that appeals to middle-aged women will also be attractive to younger men.

That's the key problem with personalization – it requires people. There's no getting around it. So remember that the next time you buy a computer book online and it suggests a Grateful Dead album to go with it. Somebody had to decide they go together. ☘

sean@sys-con.com

AUTHOR BIO

Sean Rhody is editor-in-chief of Java Developer's Journal. He is also a respected industry expert and a consultant with a leading Internet service company.

JAVA DEVELOPER'S JOURNAL

EDITORIAL ADVISORY BOARD

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

EDITOR-IN-CHIEF: SEAN RHODY
EXECUTIVE EDITOR: M'LOU PINKHAM
ART DIRECTOR: ALEX BOTERO
MANAGING EDITOR: CHERYL VAN SISE
EDITOR: NANCY VALENTINE
ASSOCIATE EDITOR: BETTY LETIZIA
ASSOCIATE EDITOR: JAMIE MATUSOW
EDITORIAL INTERN: SUZANNE AUGELLO
EDITORIAL CONSULTANT: SCOTT DAVISON
TECHNICAL EDITOR: BAHADIR KARLUV
PRODUCT REVIEW EDITOR: ED ZEBROWSKI
INDUSTRY NEWS EDITOR: ALAN WILLIAMSON
E-COMMERCE EDITOR: AJIT SAGAR

WRITERS IN THIS ISSUE

BILL BALOGLU, MARC CONNOLLY, SIRL DAVIS, BRADY FLOWERS,
ALEXIS GRANDÉMANGE, SAMUDRA GUPTA,
KEVIN LYNCH, MANI MALARANNAN, KEN MOLAY,
PAUL MOKON, BILLY PALMIERI, SEAN RHODY, AJIT SAGAR, VAUGHN VERNON,
DON WALKER, ANDREW WEBB, ALAN WILLIAMSON

SUBSCRIPTIONS

FOR SUBSCRIPTIONS AND REQUESTS FOR BULK ORDERS,
PLEASE SEND YOUR LETTERS TO SUBSCRIPTION DEPARTMENT

SUBSCRIPTION HOTLINE: 800 513-7111

COVER PRICE: \$4.99/ISSUE

DOMESTIC: \$49/YR. (12 ISSUES) CANADA/MEXICO: \$69/YR.
OVERSEAS: BASIC SUBSCRIPTION PRICE OF \$49 PLUS \$60 FOR AIRMAIL DELIVERY
(U.S. BANKS OR MONEY ORDERS). BACK ISSUES: \$12 EACH

PUBLISHER, PRESIDENT AND CO-
VP. PRODUCTION:

SENIOR VP. SALES & MARKETING: CARMEN GONZALEZ
VP. SALES & MARKETING: MILES SILVERMAN

ADVERTISING ACCOUNT EXECUTIVE: RONALD J. PERRETTI
ASSISTANT CONTROLLER: JUDITH CALMAN

CREDIT & COLLECTIONS: CYNTHIA OBIDZINSKI
ACCOUNTS PAYABLE: JOAN LAROSE

ADVERTISING ACCOUNT MANAGERS: ROBYN FORMA
MEGAN RING

ASSOCIATE SALES MANAGERS: CARRIE GEBERT
CHRISTINE RUSSELL

SALES ASSISTANT: ALISA CATLANO

VICE PRESIDENT, CIRCULATION: AGNES VANEK

CIRCULATION MANAGER: CHERIE JOHNSON

ASSOCIATE ART DIRECTOR: DINIA ROMANO

ASSISTANT ART DIRECTORS: CATHRYN BURAK
LOUIS F. CUFFARI

GRAPHIC DESIGNER: ABRAHAM ADDO

GRAPHIC DESIGN INTERN: AARATHI VENKATARAMAN

WEBMASTER: ROBERT DIAMOND

WEB DESIGNERS: GIWA ALAYIAN
STEPHEN KILMURRAY

WEB DESIGNER INTERN: PURVA DAVE

SYS-CON EVENTS MANAGER: ANTHONY D. SPITZER

EDITORIAL OFFICES

SYS-CON MEDIA, INC., 135 CHESTNUT RIDGE RD., MONTVALE, NJ 07645

TELEPHONE: 201 802-3000 FAX: 201 782-9600

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., 135 Chestnut Ridge Road, Montvale, NJ 07645.

Periodicals postage rates are paid at

Montvale, NJ 07645 and additional mailing offices.

POSTMASTER: Send address changes to:

JAVA DEVELOPER'S JOURNAL, SYS-CON Publications, Inc.,
135 Chestnut Ridge Road, Montvale, NJ 07645.

© COPYRIGHT

Copyright © 2001 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.

WORLDWIDE DISTRIBUTION BY

CURTIS CIRCULATION COMPANY

730 RIVER ROAD, NEW MILFORD NJ 07646-3048 PHONE: 201 634-7400

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. All brand and product names used on these pages are trade names, service marks or trademarks of their respective companies.



WRITTEN BY KEVIN LYNCH



Macromedia and Java: Serving the Best User Experience

Macromedia's mission is to improve the user experience on the Web. While it's best known for its Web authoring and media playback solutions, it's also committed to the viability and importance of Java as a platform for servers and has invested heavily in Java technology for two of its products, Dreamweaver UltraDev and Generator. Macromedia feels that the integration of authoring, playback, and server capabilities is key to creating the best user experience.

Dreamweaver UltraDev is the first Web development solution to enable the visual authoring of dynamic applications using JavaServer Pages. Generator, a server-side solution for delivering dynamic visual content, is built on, and can be extended, using Java.

UltraDev builds on the core architecture of Dreamweaver and adds intuitive application development features for Java developers. The product was created to be sensitive to the needs of developers. It not only protects and preserves the integrity of your application code, but can also be taught to write code the way you prefer. For most developers the underlying code is as important as the visual impact the site has on visitors. UltraDev enables developers to concentrate on delivering a great user experience by easily connecting back-end code to front-end design. It lets JSP developers take full advantage of JavaBeans authored by other team members, enabling them to tweak the final design without breaking the underlying code.

The program itself has resources to make Java developers even more productive. UltraDev provides context-sensitive JSP code reference materials within the product through a partnership with Wrox Press and its *Professional JSP* book. Developers can, for example, high-

light an object in their JSP code, then click a button. The reference material for that object will open up in the code reference panel. Developers can also extend the functionality of their JSP pages by creating reusable JSP scripts with the product's server behavior builder, teaching UltraDev to code the way they do.

The JSP reference materials, as well as many JSP scripts created or modified with UltraDev, are available on the Macromedia Exchange for Dreamweaver UltraDev, a community Web site that enables developers to extend the functionality of their product through extensions written by developers and other third parties. Since its introduction in April, the Macromedia Exchange has enabled more than 350 available extensions to be downloaded more than 2,000,000 times by Web professionals.

Generator is an enterprise server solution for producing, delivering, and personalizing real-time visual Web site graphics. It's used by leading e-businesses such as Forbes.com, Compaq, Ford.com, OpinionLab, ru4, and Hallmark.com. These companies use Generator to separate design from content to deliver visually rich information that can be easily updated.

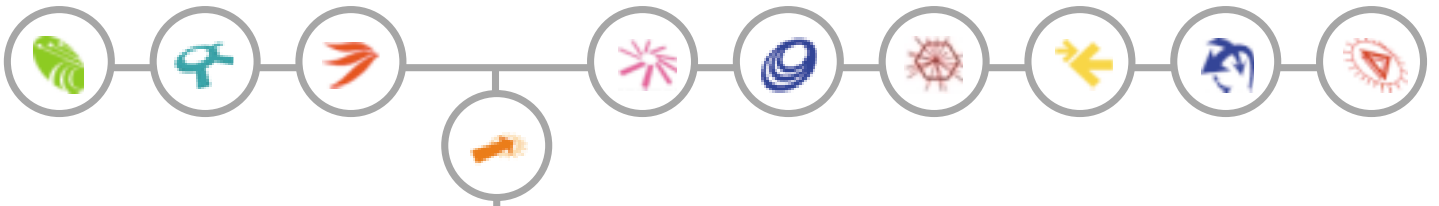
Macromedia Flash is the authoring environment for building Generator templates. The Flash Player is installed on 96% of Web desktops; developers can deploy Flash content and be assured that the largest installed base of any Web technology is able to experience it instantly. Through a wide variety of Generator objects, developers can choose the perfect visual display of information for their users. Data can be presented in scrolling lists, charts, graphics, tables, and a variety of graphic formats. The Generator server allows developers to quickly and reliably process, composite, and build Web

-continued on page 36

pr@macromedia.com

AUTHOR BIO

Kevin Lynch is president of Macromedia Products. He joined Macromedia in 1996 and has been instrumental in forming its Web strategy. As president of products, Kevin is responsible for developing Macromedia's award-winning family of software and solutions.



JAVA & macromedia

a perfect match

written by **SAMUDRA GUPTA**

How many times have we pulled out our hair trying to find a proper way to deploy high-end graphics and animation over the Web? The answer is perhaps a bit embarrassing. In DHTML concepts, with the help of JavaScript and layered components, we could render interactivity with graphics and produce some animation effects, but those were far from what we desired and what existing multimedia packages could offer for PC-based games and animation programs. By the time Java came into the picture it offered graphics-handling features, which perhaps put a ray of hope into the developer's world. Despite the fact that Java could handle graphics, it was to a limited degree and had inherent problems with graphics rendering, such as flickering, and frequent repaint problems. It was (should I say is?) a head-breaking task to write extra code to avoid those problems. Moreover, in a browser environment Java applets took a fairly long time to download and display heavy graphics, calling into question their potential as a solution.

Then Macromedia came up with award-winning products such as Flash, and developers could at last find a real solution to the problem. The Macromedia product families used vector-based graphics and framed animation and successfully produced animation in a compressed format called *Shockwave*, which had considerably smaller files and took less time to download into the client browser. So with all their rich features, Macromedia products immediately became the most popular way to present graphics and animation over Internet browsers.

Director: The Choice

Macromedia Director belongs to a similar line of products and is the tool of choice for legions of Web and multimedia developers. With Director we can create movies for Web sites, kiosks, and presentations. Movies can be as small and simple as an animated logo or as complex as an online chat room or game. Director movies can include a variety of media, such as sound, text, graphics, animation, and digital video. A Director movie can link to external media or be one of a series of movies that refer to one another.

We can view Director movies in one of three ways:

1. In the Shockwave movie format, which plays in Shockwave-enabled Web browsers
2. In a projector, which plays on your user's computer as a stand-alone application
3. In the form of a Java applet

Even though Director movies and Shockwave animation became popular and efficient ways to deploy graphics and animation over the Web, cross-platform compatibility was restricted to Windows and Macintosh environments. Director needed a way in which browsers could display movies with the help of a plug-in.

Director then offered cross-platform compatibility through its internal conversion engine, called *Xtra*, that transformed Director movies into Java applets, although Java applets aren't yet as smooth as their Shockwave equivalents. They work, however, and platform independence was achieved, making it useful for those with something other than Windows or Macintosh. The quality of graphics and the animation in the applets were dependent on the size and resolution of the images used to render the movie.

Everything is apparently set and done for a platform-independent, high-end graphics and animation deployment framework. But human beings are never satisfied, and we're still frustrated. The possibility of transforming Director movies to Java applets tempted us to apply the same rich functionality of graphics and animation outside the domain of the browser and, more important, in a platform-independent manner. Unfortunately, Macromedia didn't offer a ready-made, one-shot solution for this through Director. They offered projectors that are .exe files to run the movies as a stand-alone, but they're not platform independent.

In reality it's not too difficult to work out a solution to the problem. In fact, the bottom line is that we have to find a mechanism through which we can run the Director movie applets, loading them into a container such as Java Frame or a Java Window. Interestingly, Director allows us to produce source code of the movie applet along with a few other helper .class files to run it. We can manipulate the applet source file depending on our need and even embed our own objects that may in turn be responsible for activities such as database access, complex networking, or even RMI. In the end what we have is a powerful mechanism to build a truly platform-independent graphics and animation framework.

Creating the First Applet

Creating the applet from a Director movie is a trivial task. Any basic Director movie can be saved as a Java applet by going to the "File" menu, then choosing the "Save as Java" option. If this option isn't visible, download the "Save as Java" Xtra from the Macromedia site and install it on your machine. The options are to save it as compiled Java or as Java source code. If we choose to save it as a compiled Java source, then

Director produces all the required classes to run the movie as an applet and a .djr file of the movie that's loaded by the applet, as well as an .html file that can be loaded in a browser or applet-viewer utility to see it running.

When we want to customize the applet on our own, we must save the movie by selecting the option "Save as Source." Although things seem to be straightforward (indeed they are), it's unwise to assume at an early stage that everything within a Director movie can be converted to an applet. In fact, there are limitations (we'll discuss them in due course). But to be optimistic, we can reasonably say that a Director movie, with simpler event handling and animation features, can more or less be converted to an applet. Before covering the applet issues, it's worth discussing Director terminology and looking into Lingo, the scripting language of Director and the basic architecture of the conversion of Director movies into Java applets.

Some Useful Director Terminology

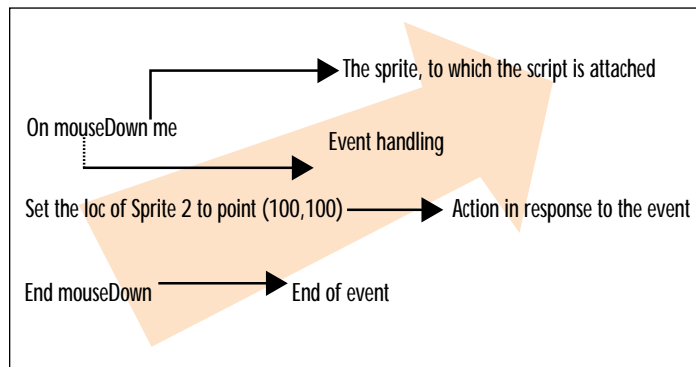
- **Stage:** The visible portion of the movie in which media elements appear.
- **Cast member:** The prepared form of any media element to be used in the Director movie. Any media elements we use have to be present in the cast window.
- **Sprite:** An object that controls when, where, and how cast members appear in a movie. Multiple sprites can use the same cast member. It's also possible to switch the cast member assigned to a sprite as the movie plays.
- **Behavior:** A prewritten Lingo script we use to provide interactivity and add interesting effects to a movie.

Introduction to Lingo

Lingo is Director's scripting language and a 4GL language with a lot of power. It offers:


- Event-handling capability
- Data parsing and manipulation ability
- Audio and video handling features
- Database interaction
- Some networking
- XML support

Everything we draw on a Director movie stage becomes a sprite, and the object of the movie animation is to control these sprites. For example, suppose we draw a button (Sprite1) on the stage, and by clicking it we want to move another Sprite, a ball (Sprite 2), to point 100,100. The equivalent expression in Lingo would be:



As mentioned, the Director movie is a kind of framed animation, and each sprite on the stage is assigned several behaviors. We can control these behaviors with the help of Lingo and, loosely speaking, that's the key to Director movie animation.

It's possible to write two kinds of scripts with Lingo. A frame script controls the behavior of the movie when it reaches a particular frame. A



behavior script controls the sprite's behavior, such as dancing and movement. By attaching custom scripts to the sprites and cast members, we can achieve a high level of framed animation. It's recommended that the developer consult a thorough tutorial on Lingo before developing complicated movies in Director.

built-in cast member types – shape, sound, bitmap, field, transition, and script. We can create custom subclasses of Member.

The Member class's public API primarily provides get-and-set access to Lingo-visible properties, such as getWidth() and setText(String). The DirectorMovie.getMember() functions fetch an existing member object given its name or number. Use the Sprite.getMember() function to fetch the cast member attached to a particular sprite.

How the Movie Works as an Applet

The basic architecture of how the applet reproduces the movie can be summarized in the following manner.

Every running applet has exactly one instance of movie-Name.class, which is the first object created when the applet runs. This object reads the media file, creates the score and cast data structures, then begins to play the movie. According to the movie's tempo (the number of frames covered in a second), this object periodically advances the frame counter, dispatches frame events to active sprites and their behaviors, and redraws the stage. As the applet receives mouse and key events, this object dispatches the events to movie, sprite, and cast member scripts. The DirectorMovie class's public API supports Lingo commands that manipulate the entire movie and access to sprite and cast member objects.

Standard Java Classes

Director's "Save as Java" Xtra can minimize the player to reduce the size of the applet. If Xtra doesn't minimize the player, it contains standard player classes. A minimized player contains only those classes the converted movie requires. The included classes contain the code only for the features the applet uses.

Core Classes

Following are the key classes for the Director movies running as applets. These classes represent a movie's basic structure. All classes aren't always required to run the movie applets. If the movie doesn't use any of the features available in the Lingo value or helper classes, they can be omitted and the minimized player would contain only the core classes. Each class provides an API through which we control the behavior of the objects of the corresponding classes.

DirectorMovie (extends Applet)

Contains all data and functions that pertain to the overall movie, such as:

- The score
- The cast, which is a list of objects of the Member class and can represent all the sprites taking part in the movie
- Movie properties, such as the frame counter, list of active sprite objects and Lingo commands for common features, such as network operations, mouse and key interactions, and movie control
- Event dispatching
- Interacting with the browser (via overridden applet methods)
- Handling media files

DirectorMovie is an abstract class. The class movieName derives from the DirectorMovie class, where movieName represents the movie's final name. For example, a movie named testMovie will have the corresponding class testMovie.class.

Member

In a movie converted to Java every cast member (either taken to the stage or not) has a corresponding member object that's created when the movie begins playing.

A Member object contains the data and functions required to load, draw, and use a single cast member. The Member class handles all the

“by attaching custom scripts to the sprites and cast members, WE CAN ACHIEVE A HIGH LEVEL OF FRAMED ANIMATION”

Sprite

Every sprite that's currently active has one instance of the sprite class, which the player creates and destroys as required.

A sprite object contains the data and functions required to animate a single sprite, including stepping and interpreting the score and dispatching events to behaviors. The sprite class handles all built-in sprite types, and we can create custom subclasses of sprites.

The sprite class's public API primarily provides get-and-set access to Lingo-visible properties, such as getInk() or setLoCH(). We can use the DirectorMovie.getSprite() function to fetch a sprite object by number and manipulate the sprite's properties by the public API of the sprite class. The Behavior._s property is a reference to the sprite object that a particular behavior is attached to.

Behavior

This abstract class is the base class for all sprites and frame scripts that contain property declarations. Each such sprite or frame script is a unique subclass of behavior. For sprite and frame scripts that contain property declarations, the behavior object's public API consists of the following:

- **Event-handling methods:** All handlers in other sprite and frame scripts are grouped into the single function .uberHandleAnEvent(). Grouping these handlers reduces the number of distinct classes, which substantially reduces the player's size.

Lingo Value Classes

When the export Xtra for Java translates scripts to Java, it tries to determine whether a variable or parameter's data type is one of Java's built-in data types: int, String, float, or Boolean. If the Xtra can determine that the variable or parameter is one of Java's built-in data types, it takes the same data type in Java. Sometimes the Xtra can't determine the data type.

For example, several set statements assign a variable different data types if the type isn't a built-in Java data type. In these situations the Xtra gives the variable or parameter the Java type LVal, or "Lingo value." An LVal object can be a null, integer, string, double, sprite, member, symbol, linear list, property list, rect, or point data type. Various subclasses of LVal handle the possible types of data that an LVal object can hold. The player creates and destroys Lingo values as required. After Java creates a Lingo value object, the object can't change its type.

- **LVal:** The base class of all Lingo values. Its public API contains all the Lingo-visible operations on any valid data types. These operations include fetching the value as a certain type and all list and arithmetic operations.
- **LPoint:** Extends LVal and implements Lingo values that are points. We can use LVal.accessProp() to access the point's locH and locV properties.
- **LRect:** Extends LVal and implements Lingo values that are rects. We can use LVal.accessProp to access the left, right, top, bottom, width, and height of the rect.

- **LList:** Extends LVal and implements Lingo values that are linear lists. We can use various functions of the LVal class to access the list.

Helper Classes

These classes have no public API. The Xtra includes them in the player when they're required.

- **NetworkFetch:** This class implements getNetText() and associated functions.
- **J10IS, J11IS, MyMemoryImageSource:** These classes help decode GIF and JPEG images.

Classes Generated by the Export Xtra for Java

The Export Xtra for Java places the Java source code it generates in the file movieName.java. This file always contains the class movieName and may also contain one or more behaviorMemberName classes.

- **movieName Extends DirectorMovie:** Each Director player for Java has one subclass of DirectorMovie named after the original movie file. The Export Xtra for Java creates this class. The movieName's data and functions are simply the original Director movie's global variables and movie handlers.
- **behaviorMemberName Extends Behavior:** Every sprite and frame script that has properties is a unique subclass of behavior. Each behaviorMemberName class's data and functions are simply the properties and handlers of the original behavior script. In addition, each behaviorMemberName class also has the Member variable __m, which is a reference to the DirectorMovie object that owns the behavior.

Limitations of Custom Java

Following are some known tasks that custom Java can't do in the Director player for Java.

JavaBeans can't easily be embedded because they're drawn differently than Director cast members and sprites. To embed the bean we have to rewrite the bean's source as a class that inherits from the member and sprite.

AWT components and functionality can't be integrated because they're drawn differently than Director draws. Director's rich compositing model uses a custom Java engine that doesn't mesh with the AWT drawing model.

Java's other inherent limitations, including security restrictions and lack of direct connection with the operating system, also limit how much we can customize Java in an applet.

Not all the events that Lingo is capable of handling are converted to the applet. For example, a doubleClick event can't be directly converted to an applet.

Many animation and movie features included in Director can't be converted to Java applets.

The Framework

In the context of developing a mechanism to load and play Director movie applets, we need to understand how an applet works. While executing, applets need to have information regarding the environment in which they're running, so they require a standard way of interacting with their environments. This is provided by two interfaces: AppletStub and AppletContext, both defined in the java.applet package. When an applet is first created, a stub is attached to it using the applet class's setStub() method. This stub serves as the interface between the applet and the browser environment or applet viewer environment in which the applet is running. The AppletContext interface corresponds to an applet's environment – the document containing the applet and other applets (if any) in the same document. The methods in this interface can be used to obtain information about its environment.

We'll create the applet, provide a stub and context for it, place it in a container to be displayed, and execute its actions and properties. This is

possible only if the applet is the kind of component that can be placed inside a container. If we look into the Java class hierarchy, we see that the applet class inherits from the panel class that in turn is inherited from the component class (see Figure 1).

An applet also inherits the properties of a component, and it should be possible to place it inside a standard Java container, such as a frame or window, as simply as any other graphics component, like Button or TextField. So far it's as simple as that. But applets are a special type of component in the sense that they're not static like AWT components, but possess their own life cycles to execute. At the same time they can also act as a container that can be a placeholder for other AWT components.

In a nutshell, to design and implement a framework for playing the Director movie applets as stand-alone applications we need to achieve the following:

- Get the Director movie in the form of an applet that's like any other normal applet extending from the java.applet.Applet class.
- Implement a class loader that will load a named applet class.
- Provide the required AppletStub and AppletContext interfaces to the applet.
- Implement a container object, typically a frame or window, to hold the applet.

Designing the ClassLoader

The CustomClassLoader class extends from the java.lang.ClassLoader, an abstract class that contains an abstract method called loadClass (String className, Boolean resolve) (see Listing 1). If the resolve flag is true, the method should call the resolveClass() method of the resulting Class object. Our CustomClassLoader class implements this

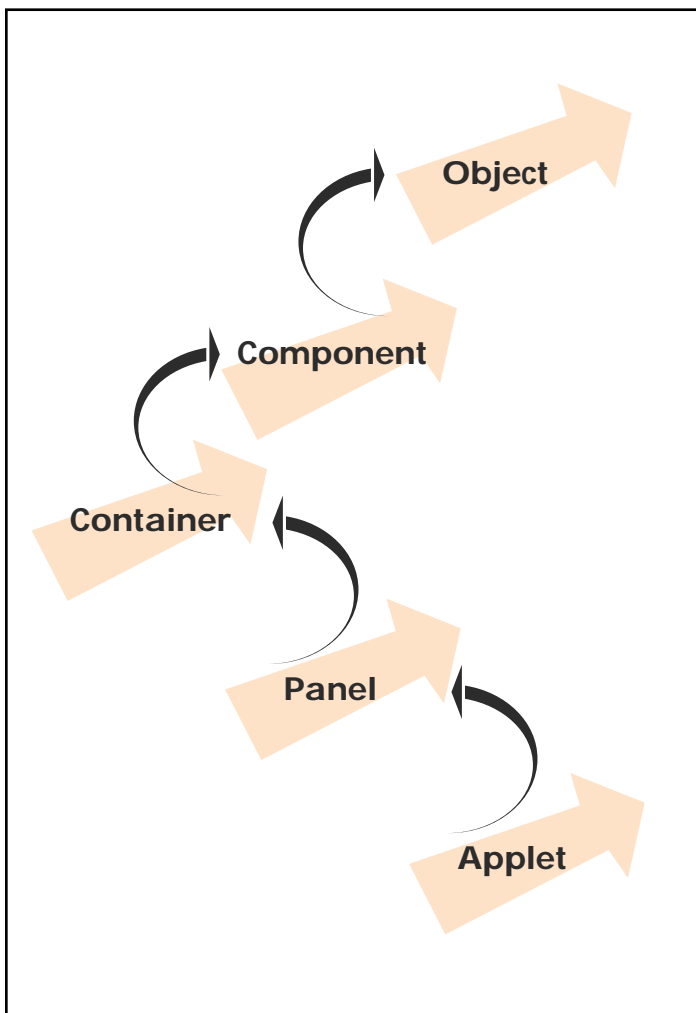


FIGURE 1 Class hierarchy of the java.applet.Applet class

method and returns the Class object specified in the variable className. Following is the code snippet doing the required job:

```
.....
Hashtable classDefs=new Hashtable();
.....
public Class loadClass(String className, boolean flag) {
    try{
        if(flag) {
            resolveClass(findSystemClass(className));
            classDefs.put(className, findSystemClass(className));
        }
    }catch(Exception e){ }
    return (Class)classDefs.get(className);
}
```

The advantage of putting the resolved class into a hashtable is that the same class doesn't get loaded twice; if it's already been loaded, it's fetched from the hashtable and returned as the class object.

The CustomStub class implements both AppletStub and AppletContext interfaces defined in the java.applet package and provides the custom implementation of the methods defined within those two interfaces (see Listing 2). In our program the CustomStub constructor expects movieName to be passed because the movie applet queries the getParameter(ShockwaveMediaFile) about the .djr file to be loaded. This is normally passed via the PARAM tag of the APPLET tag in the HTML file. To return a proper .djr file name, the CustomStub needs to know the name of the movie, which it does via the constructor.

The next step is to implement a movieLoader class that's also a container and can contain an applet and attach the necessary CustomStub object to the applet (see Listing 3). This class essentially makes a call to the loadClass() method of the CustomClassLoader class, passing the name of the movie applet. It's important to note that loading the applet doesn't mean the applet is executing its life cycle. We have to explicitly call the life cycle methods of the applet (init(), start(), destroy()) as needed and provide it a CustomStub object to see it perform the actions defined within them. The following lines of the movieLoader class do precisely that.

```
CustomClassLoader loader=new CustomClassLoader();
CustomStub stub=new CustomStub(movieName);
Applet theApplet=(Applet)loader.loadClass(movieName, true);
theApplet.setStub(stub);
theApplet.init();
theApplet.start();
```

The overall proposed architecture is presented in Figure 2.

Live Example

Once these classes are implemented we have everything in place to load and run the movie applet, so it's time to build a live example of the concepts explored. I'll present a bare-bones example of the process that includes our own Java objects and methods within the movie applet. It minimally illustrates Director's animation capability and how we can integrate the Director movie into our other external Java programs. The example will obtain a random number between one and six from the NumberGenerator class and show the image of the generated number on the stage in response to clicking on a button.

The Movie

To produce the movie (sample.djr), I've created the .bmp images of numbers one to six, made them cast members, drawn them onto the stage to make them available as sprites, and put them on the stage so they become invisible at the start of the movie. I'll pick the relevant

sprite that corresponds to the number obtained from the NumberGenerator class and place it onto the stage at a particular location, 250,100, in this case.

The NumberGenerator Class

This is a simple class that generates a random number (see Listing 4). It has a default constructor and a method called generateNumber() that returns a random integer between one and six. The movie applet will instantiate this class and call the generateNumber() method each time we hit a button on the stage. The sprite corresponding to the generated number is then displayed to the location (250,100). This sounds much simpler and may be frustrating for our animation lovers, but it teaches us a few very basic rules of creating and integrating a Director movie applet with other external Java programs.

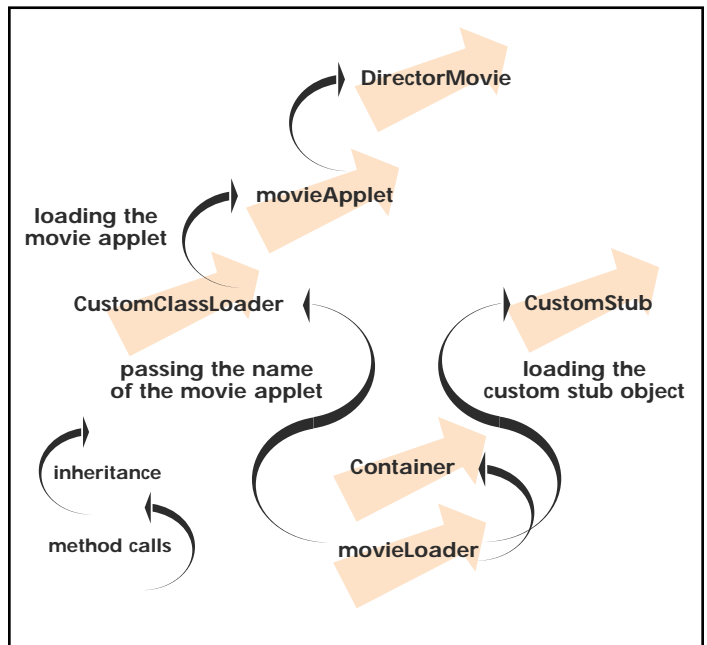


FIGURE 2 Object relationship of the developed framework

Creating the Movie Applet

It's beyond the scope of this article to show every step needed to follow a Director movie (which readers can learn from the tutorial inside the Director package itself). For this example, I've drawn a button on the stage and attached a dummy script to the button that looks like this:

```
On mouseDown me
    set the location of Sprite(1) to point(250,100).
End mouseDown
```

The script typically handles a mouseDown event on the button and brings any particular sprite to the determined location (250,100). If we don't attach any script to the button before converting it to the Java applet, the button sprite won't be configured to register any event through the Java code. This occurs because all the behaviors of the sprites are registered inside the behavior class. Each member of the movie has to be notified in a way that it's capable of handling an event on it. This is according to Macromedia's model of handling the movies as Java applets.

Having done that, if we save the movie as the source Java file (from the "Save as Java" option), we'd see the sections of the code shown below.

The movie class (in this case I've named the movie as Sample) extends the DirectorMovie class, which in turn extends the Applet class (see Listing 5).


```

public class sample extends DirectorMovie
{
public sample()      {
    __m = this;
}
static public sample __m;

```

Following is an event-handling method that typically holds the translation of the movie scripts:

```

// --- Movie Script Translations ---

public void uberHandleAnEvent (Member member, int eventCode)
{
if ( member.castNumber == 1 && member.memberNumber == 8 ) //Translation of script: castMember8
{
    switch( eventCode )
    {
case DirectorMovie.__MouseUp: // mouseUp
{
//set the loc of Sprite(1) to point (250,100)
__m.getSprite(1).setLoc(new Lpoint(250,100).asPoint());
break;
}
default:    break;
}
}
}
}

```

The event handling contains the Java equivalent translation of the Lingo script we attached to the button. We need to modify this code to our specific needs to show the generated number sprite on the stage.

Adding Our Own Objects

Now we're ready to add our own NumberGenerator object to the movie to obtain the random number between one and six generated by the NumberGenerator class. First we declare a NumberGenerator object like this at the class level:

```
NumberGenerator generator= null;
```

Then we initialize the object in the constructor of the sample.java.

```
generator=new NumberGenerator();
```

Now we need to write the code to obtain the number by invoking the generateNumber() method of the NumberGenerator class. We do this in the event-handling portion of the sample.java and bring the corresponding sprite to the predetermined point (250,100) location (see Listing 5). Each time we click the button, a number is generated and shows the relevant sprite. But remember we also have to remove any previous sprite present on the stage.

There are two ways to do this:

1. Read the number generated before generating a new number, and remove the corresponding sprite from the stage.
2. Before generating a new number or after obtaining the new number, remove all the number sprites from the stage, then show the sprite corresponding to the generated number.

Since the first method seems more efficient to me, I'll follow that.

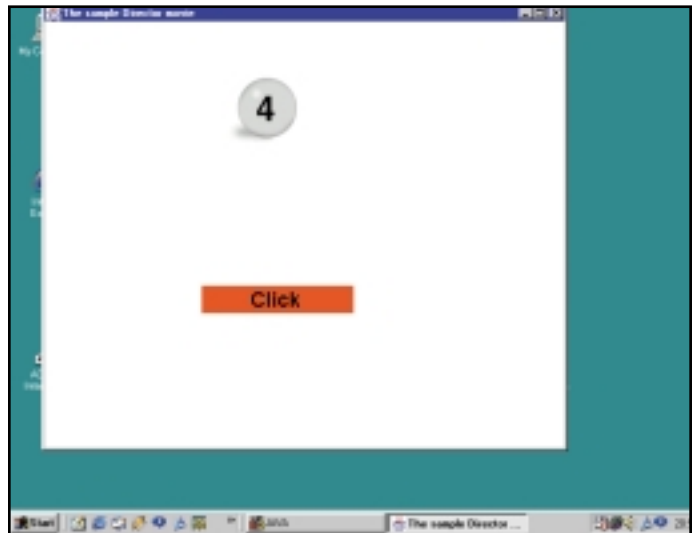


FIGURE 3 Screenshot of the Director movie running as Java application

Removing a sprite from the stage is as simple as moving it to a high location, for example 2000,2000. This is no doubt beyond the stage of the movie, which is designed to be full-screen size in any resolution.

Setting the Classpath

We must include the current working directory in our classpath or mention it explicitly while executing the movie program.

Running the Application

We've completed the framework, so it's as simple as running the application from the command line by typing the command:

```
java movieLoader sample
```

and the movie appears (see Figure 3). We can generate and see the number by clicking on the button on the stage.

Conclusion

This article is a guideline to running Director movies as stand-alone Java applications. Keep in mind that this is a developing field and improvements are subject to further research. Take interest in this area and develop more robust and efficient frameworks to work in.

Resources

Additional information is available at www.mcli.dist.maricopa.edu/director/javalist/dir2java.html.

For Java support within Lingo, visit www.macromedia.com/support/director/how/expert/javasupport/Javasupp.html ☛

AUTHOR BIO

Samudra Gupta has been involved in the Java field in various research projects for more than three years. Currently, he works as a consultant to various UK concerns for their intranet- and Internet-based e-com-

samudrag@hotmail.com

“this is a developing field and IMPROVEMENTS ARE SUBJECT TO FURTHER RESEARCH”

Listing 1 CustomClassLoader.java

```
/**
 * this class is a customized class loader which loads a named
 class and puts into a Hashtable for future reference
 */
import java.io.*;
import java.util.*;

public class CustomClassLoader extends ClassLoader
{
    Hashtable classdefs; //Hashtable for storing the reference
of the loaded class(s)

    public CustomClassLoader()
    {
        classdefs=new Hashtable();
    }

    /**
 * this method loads a named java class file and returns it as a
 Class object
 */
    public Class loadClass(String className, boolean flag)
    {
        try
        {
            if(flag)
            {
                resolveClass(findSystemClass(className));
            }

            //putting the class name into the hashtable
            classdefs.put(className, findSystemClass(className));
        }
        catch(Exception e)
        {
            System.out.println("Error in loading the class :
"+e.toString());
        }

        //returning the class by picking up from the hashtable
        return (Class)classdefs.get(className);
    }
}
```

Listing 2 CustomStub.java

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

/**
 * This is a custom class providing the required AppletStub and
 AppletContext environment to the
 * movie applet. This is a very minimal implementation of the
 AppletStub and AppletContext interfaces with
 * only the methods implemented as they are required by the movie
 applet.

 * The getCodeBase() , getImage() and getParameter() methods are
 used by the movie applet.
 * So they are implemented accordingly.
 */

public class CustomStub implements AppletStub, AppletContext, Enum-
eration
{
    private URL codeBase;
    private URL documentBase;
    private String mName=null;
    /* The constructor accepts the name of the movie because from the
 getParameter() method we have to return
 * the movieName.djr as a string which is used by the movie applet
 to display the movie.

 */
    public CustomStub(String movieName)
    {
        mName=movieName;
        try
        {
```

```
        codeBase=new URL("file:"+System.getProperty("user.dir")+"/");
        documentBase=codeBase;
    }catch(Exception e)
    {
    }
}

public void appletResize(int w, int h){ }

public AppletContext getAppletContext()
{
    return (AppletContext)this;
}

public URL getDocumentBase()
{
    return documentBase;
}

public URL getCodeBase()
{
    return codeBase;
}

public String getParameter(String param)
{
    return mName+".djv";
}

public boolean isActive()
{
    return true;
}

//AppletContext methods

public Applet getApplet(String name)
{
    return null;
}

public Enumeration getApplets()
{
    return (Enumeration)this;
}

public AudioClip getAudioClip(URL url)
{
    return null;
}

public Image getImage(URL url)
{
    return Toolkit.getDefaultToolkit().getImage(url);
}

public void showDocument(URL url){ }
public void showDocument(URL url, String target){ }
public void showStatus(String status){ }

//Enumeration methods

public boolean hasMoreElements()
{
    return false;
}

public Object nextElement()
{
    return null;
}
}
```

Listing 3 MovieLoader.java

```
/**
This class loads a Director movie applet and displays in a java
Frame component
*/

import java.applet.*;
import java.awt.*;
```

```

import javax.swing.*;

public class MovieLoader extends JFrame
{

    CustomClassLoader loader=null;
    Class theClass=null;
    Applet theApplet=null;
    CustomStub stub=null;

    public MovieLoader()
    {
        super("The sample Director movie");

        try
        {
            //setting the size of the frame to 400X400
            this.setSize(400,400);

            //setting the background color of the frame to white
            this.setBackground(Color.white);

            //making the frame visible
            this.show();

        }catch(Exception e)
        {
            System.out.println(e.toString());
        }

    }

    /**
    * this method loads a named movie applet
    */
    public void loadMovie(String movieName)
    {
        try
        {

            //initializing a CustomClassLoader object
            loader=new CustomClassLoader();

            //loading a named movie applet class through the CustomClass-
            Loader object
            theClass=loader.loadClass(movieName,true);

            //casting the loaded class to an Applet class
            theApplet=(Applet)theClass.newInstance();

            //initializing the CustomStub class with the name of the movie
            applet
            stub=new CustomStub(movieName);

            //providing the applet a CustomStub object
            theApplet.setStub(stub);

            //setting the width and height of the applet
            theApplet.setSize(400,400);

            //adding the applet to the frame
            this.getContentPane().add(theApplet);
        }catch(Exception e){ System.out.println(e.toString());}

        //calling the life cycle methods of the applet

        theApplet.init();
        theApplet.start();
        validate();

    }

    public static void main(String args[])
    {

        new MovieLoader().loadMovie(args[0]);
    }
}

```

Listing 4 NumberGenerator.java

```

class NumberGenerator
{

```

```

    public NumberGenerator()
    {

    }

    public int generateNumber()
    {
        java.util.Random random = new java.util.Random();
        int i = random.nextInt(5)+1;
        return i;
    }
}

```

Listing 5 sample.java

```

import java.applet.Applet;
import java.awt.image.*;
import java.awt.*;
import java.util.*;
import java.net.*;

public class sample extends DirectorMovie
{

    NumberGenerator generator = null;
    int number = 0;
    int usedNumber=0;

    static public sample __m; //global movie variable declaration

    public sample()
    {

        __m = this;

        generator = new NumberGenerator();

    }

    /**
    * --- Movie Script Translations ---
    */

    public void uberHandleAnEvent (Member member, int eventCode)
    {

    if ( member.castNumber == 1 && member.memberNumber == 8 ) //ans-
    lation of script: castMember8
        {
            switch( eventCode )
            {
                case DirectorMovie.__MouseUp: // mouseUp
                {
                    //finding if there is a number already displayed and
                    then moving it out of stage
                    if(usedNumber != 0)
                    {
                        __m.getSprite( usedNumber ).setLoc(new LPoint(
                        2000,2000 ).asPoint());
                    }

                    //obtaining a generated number from the NumberGenerator class
                    number=generator.generateNumber();

                    // set the location of the sprite corresponding to the gener-
                    ated number
                    // to point(250,100)
                    __m.getSprite( number ).setLoc(new LPoint( 250, 100
                    ).asPoint());

                    usedNumber = number;
                    break;
                }
                default: break;
            }
        }
    }
}

```



WRITTEN BY ALEXIS GRANDMANGE

Last month in **JDJ** (Vol. 6, issue 1) we looked at the advantages of downloading servlets and JavaServer Pages (JSP) from a repository, for example, the same way a browser downloads applets. We described a simple implementation of this concept based on a servlet and a custom class loader. This tool, named *JSPservlet*, handled servlets and JSP packaged in JAR archives to minimize the number of connections and transfers required.

This month I'll show you how to publish an archive, update it or force its download through a JSPupdate servlet, and extend the solution to handle resources, HTTP caching, request forwarding, page inclusion, and JSP beans. The code listings for this article can be found on the **JDJ** Web site, www.JavaDevelopersJournal.com.

Archive Update

This simple JSP, JSPupdate, handles archive publishing and updates (see Figure 1).

JSPservlet and JSPupdate are packaged in a Web application, typically in a WAR archive that's described by a web.xml deployment descriptor (see Listing 1). This archive is a general-purpose agent responsible for downloading the target presentation archives and routing requests to their servlets and JSPs.

To publish a new archive you must query the proper agent and provide the archive name and the

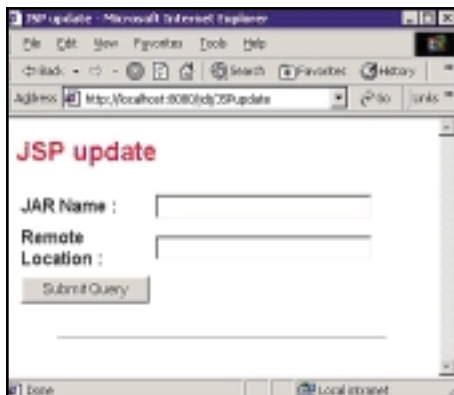


FIGURE 1 JSPupdate display



Part 2 of 3

Develop a solution that's portable across Java application servers

remote location where it should be downloaded from. Simply identify the agent on your browser by its URL, in this example `http://localhost:8080/jdj/JSPupdate`, where `http://localhost:8080` identifies the Java server and `jdj` the agent's Web application. This displays the form in Figure 1. Fill it and click the button to start the publishing. Use the same form to change the archive location or to force a new download. In the latter case you don't even need to fill the remote location.

Let's go back to the tool design and walk through the implementation of JSPupdate (see Figure 2).

Tool Design

The JSPservlet is a special servlet that handles HTTP requests for a Web application and forwards them to target servlets and JSPs with the help of the following objects:

- **JSPHandler:** Manages Web applications and maintains a ClassEntry map.
- **ClassEntry:** Manages archives and maintains a cache of target objects.
- **JSPloader:** Maintains a cache of target classes. Therefore JSPupdate handling implies the

following steps:

1. Identify the relevant JSPHandler and create it on the fly if it doesn't exist.
2. Find the ClassEntry responsible for the archive.
3. If it doesn't yet exist, create a ClassEntry. In this case the tool acts as an archive publisher.
4. Otherwise, unreference the JSPloader, clear the target object cache, and instantiate a new JSPloader.

The first step is implemented in JSPupdate and JSPHandler. Listing 2 shows the JSPupdate code. I prefer to use the GET mode to simplify updates by programs or scripts. I'll return to this point in the next section. In the script, starting on line 27 in Listing 2, I first get the JAR name you filled on the form and the application name, `contextPath`, from the URL. Then I look for the corresponding JSPHandler in the JSPHandler's HashMap, and finally I invoke the JSPHandler's `update()` method. I postpone the explanation of the case in which the appropriate JSPHandler doesn't exist to the RequestDispatcher section.

Listing 3 shows the JSPHandler.update() implementation. Remember the tool mini-

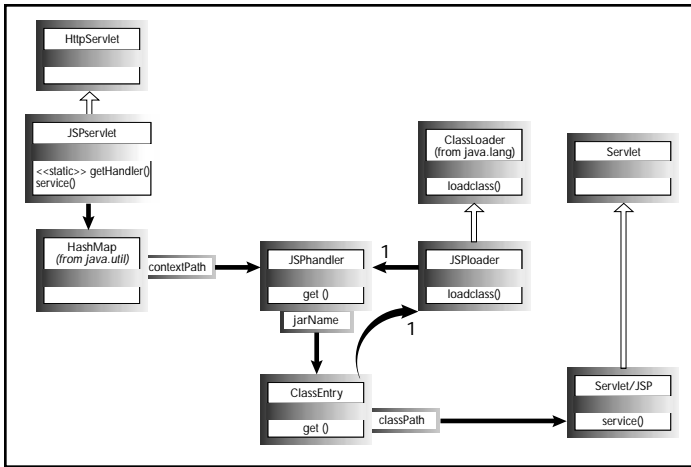


FIGURE 2 Tool class diagram

mizes downloads from a central repository and handles its unavailability thanks to a local archive copy. `JSPHandler.update()` first removes this archive cache with `File.delete()`. Then, if you filled the Remote Location field, it updates the `remoteLocProp` property and persists it on `remoteLocFile`. Eventually `JSPHandler.update()` looks for the appropriate `ClassEntry` in the `classEntries` `HashMap` and invokes its `update()` method. If it doesn't find it, it creates a new `ClassEntry` and adds it to `classEntries`.

Listing 4 shows the `ClassEntry.update()` implementation. It first invokes the `destroy()` method of all cached target objects. Then it clears `servletObjects`, the target object's `HashMap`, to unreferenc them and then unreferences the corresponding `JSPloder` instance. Next `ClassEntry.update()` invokes the garbage collector, which can free the target and `JSPloder` objects and also the target classes and static data. Though the garbage collection can take time, it reduces the Java server footprint and improves its behavior. I considered the garbage collection duration to be a minor drawback as I designed `JSPupdate` to be invoked outside peak hours. Once the Java Virtual Machine (JVM) reclaimed the memory occupied on behalf of the archive, `ClassEntry.update()` created a new `JSPloder` and a new target object cache.

Scripted Update

`JSPupdate` uses the GET mode. To require the update of an application whose URL is `www.iamakishirofan.com/gunnm` for a JAR named `gally` stored in a repository located in `http://myserver`, you simply need to use the URL `www.iamakishirofan.com/gunnm/JSPupdate?jarName=gally&remoteLocation=http%3A%2F%2Fmyserver&Submit=JSPupdate`.

Listing 5 shows a Java class, `UpdateClient`, you can use to update an archive from the command line or from a script. To update the application above, invoke `UpdateClient` either with `Java JSPservletPkg/UpdateClient http://www.iamakishirofan.com/gunnm gally http://myserver` if you want to publish or update the remote location or with `Java JSPservletPkg/UpdateClient http://www.iamakishirofan.com/gunnm gally` if you simply need to force a download.

`UpdateClient` first builds a URL string with the `UpdateClient`'s parameters. To convert the remote location to a MIME format that's appropriate in a URL, `UpdateClient` simply uses `URLEncoder.encode()`. Then it creates a URL and opens and reads a stream, which it parses to check the Java server response. Use the exit code in your script to handle error cases; the most common one is server unavailability.

Resources

Consider the common case in which a JSP or servlet refers to an image with a URL that's relative to the current path. Since `JServlet` is configured in the Web application deployment descriptor to handle all its requests, it also has to process image requests. This case raises three issues:

1. How to detect an image request
2. Where to download the image from
3. Where to handle the request

I chose to delegate images and other resource handling to `JSPloder` because, as we'll see later in the beans section, it has to address other resource needs anyway.

Where should we download the resources from? Should we cache them? These aren't trivial issues, as an image is much larger than a JSP or servlet. My choice is to support resources that are included in the archive file or stored in the same remote location as the archive, and to cache resources in memory.

Listing 6 shows the resource handling in `JServlet`. `JServlet` detects images and other resources such as HTML files after their URL extension. It also sets the content type according to the URL extension. Then it uses `JSPHandler.getResourceAsStream()` to get an input stream on the resource from `JSPloder`. `JSPHandler` simply forwards the request to the appropriate `ClassEntry`, which invokes `JSPloder.getResourceAsStream()`. If `getResourceAsStream()` doesn't find the resource, `JServlet` invokes `HttpServletResponse.sendError(SC_NOT_FOUND)`, which builds an HTTP response with a 404 status, which indicates that the requested resource is not available. Otherwise `JServlet` reads the input stream and rewrites it on the response output stream.

To support resources embedded in an archive, I modified last month's `JSPloder.ParseStream()` method. Remember this method is invoked during `JSPloder` construction to parse the archive content that's read from the local archive cache or from its remote location. In the latter case it's also responsible for storing the archive in the local cache. However, the modification is minimal, as you can see in Listing 7. `JSPloder` maintains a resources `HashMap` that acts as a resource memory cache the way classes act as a class memory cache. `parseStream` stores a resource in a byte array in `resources`, instead of storing a class in classes.

Listing 8 shows the `JSPloder.getResourceAsStream()` implementation. It first tries to retrieve the resource from the resources memory cache. If it doesn't find it, it tries to download the resource from the same location as the archive with `URL(remoteLoc).openStream()`. Then it stores it in `resources`. If a resource is stored in the archive, it's always served from `resources`. A resource that must be downloaded is downloaded only once, then served from `resources`. If the resource is neither in the archive nor remotely available, `getResourceAsStream()` delegates the request to `getResourceForward()` in order to support local Java server resources. `getResourceForward()` first tries to find the resource in `JServlet`'s Web application using the `getResourceAsStream()` method of `JServlet`'s class loader, then tries to find it elsewhere using the `getResourceAsStream()` method of the `JSPloder` parent class loader.

Proxy and Browser Caching

Figure 3 displays a typical HTTP caching scenario with three actor types, browsers, a proxy, and an HTTP server. The first browser requires a URL that the proxy can't find in its cache. So it requests the URL from the HTTP server with an HTTP GET. The HTTP server returns a response, which also includes `Expires`, `ETag`, and `Last-Modified` header fields. The `Expires` field gives the date/time after which the response is considered stale, the `ETag` field provides a Entity tag value that can be used for comparisons, and the `LastModified` tag indicates the date and time the server believes the data was last modified.

The proxy stores the response in a cache and returns it to the browser. Then a second browser requires the same URL. The proxy finds the response isn't stalled so it returns it to the browser without requesting the HTTP server. Next, a third browser requires the same URL and this time the response is stalled but still in cache, so the proxy asks the HTTP server if the response is still valid with a conditional HTTP GET (a GET with an `IfNoneMatch` or `IfModifiedSince` field). The HTTP server checks if the Entity tag is the same in the case of `IfNoneMatch`, or if the `LastModified` tag hasn't changed in the case of `IfModifiedSince`. If yes, it sends a `NotModified`

response without a body. If the browser requested an image, the image is not transferred. If it requested a dynamic page that required RDBMS access or heavy computation, this processing is not needed.

The HTTP server sends an updated Expires value in its NotModified response, so if a fourth browser requests the same URL before the updated date/time, the proxy will again serve the response without involving the HTTP server.

I described a proxy's behavior for clarity, but a browser also caches the responses it receives and behaves exactly the same way regarding the header fields presented in Figure 3. It's the reason I had to involve four different browsers in the scenario. As a consequence, an HTTP server can drive both proxy and browser caching with the same code.

To implement that mechanism for resource requests, I had to make two decisions:

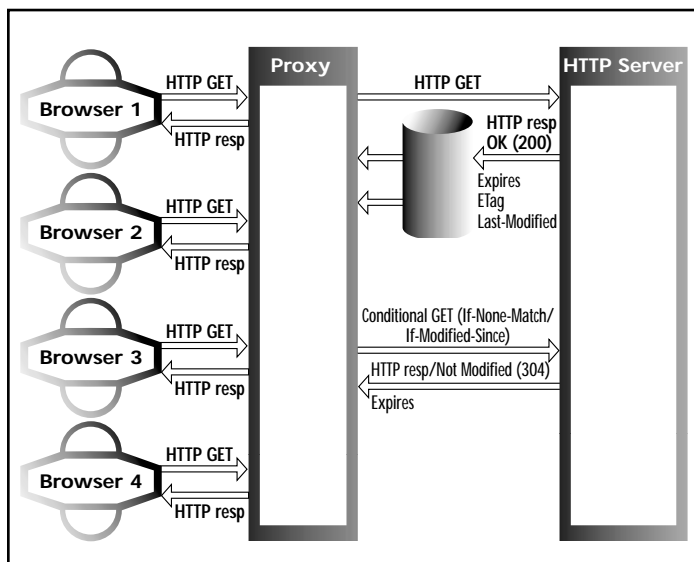


FIGURE 3 Caching mechanisms

1. Where should I take the LastModified date/time?
2. Should I implement Expire?

Remember the `JSPloader.getResourceAsStream()` implementation? It first tries to retrieve the resource from the archive, then from the same location as the archive, and eventually by asking the Java server class loader. When the resource is stored in the archive, it picks up the Last-Modified date/time from the archive entry with `JarEntry.getTime()`. When the resource is stored in the same location as the archive, it uses a `URLConnection` object to download it. `URLConnection` acts as a browser, so it has access to HTTP headers. It even provides helper methods for the most common headers, such as `URLConnection.getLastModified()` for Last-Modified that's invoked by `JSPloader.getResourceAsStream()`. In the last case where `JSPloader.getResourceAsStream()` asks the Java server for the resource, I use the archive cache creation time. The rationale is this sort of resource is typically stored on a Java server and therefore cheap to retrieve.

The bottom line is:

1. If an archive or downloaded resource hasn't changed, JSPervlet will return NotModified even after an archive update.
2. For a resource retrieved by the Java server class loader, JSPervlet will return a full response at the first request after an archive update.

When the proxy receives a response containing ETag or LastModified, it can set an internal expiration value. However, this behavior isn't mandatory and can vary, so I prefer to implement Expire and let you set it in an additional initialization parameter, `expiration`, with a five-second default. The HTTP 1.1 specification allows you to go up to one year, but its optimal value depends on your configuration. The higher it is, the more time it will take to refresh caches after an update. If your browsers are on the same LAN as the Java server, don't worry about round-trip delays.

Let's go back to Listing 6 to look at the implementation details.

JSPervlet checks if the HTTP request was conditional. More precisely, it retrieves the value of `IfNoneMatch` and `IfModified` header fields. If they're set, it checks, respectively, if client Entity tag and Last Modified date/time are still the same as server ones. If they are, JSPervlet returns an HTTP response with a status NotModified (303), using `HttpServletResponse.sendError(SC_NOT_MODIFIED)`. This response includes an Expires field set with:

```
HttpServletResponse.setDateHeader("Expires", System.currentTimeMillis() + jh.expiration * 1000)
```

`setDateHeader` is another convenient helper method that simplifies setting a date header field. It takes two parameters, the name of the field and the elapsed time since the epoch (January 1, 1970). JSPhandler computes it using JSPhandler's `expiration`, which contains the expiration initialization parameter.

If the HTTP request is not conditional or if cache entries are stalled, JSPervlet sends the resource. It's set before the `Date`, `Cachecontrol`, `LastModified`, `ETag`, and `Expires` header fields. `Date` represents the date and time the message originated. JSPervlet builds this `Date` the same way as `Expires`. `Cachecontrol:public` indicates that the response may be cached by any cache. I already covered `LastModified` and `ETag`. Both contain the date and time extracted by JSPloader. `LastModified` handling is slightly more complex, as JSPervlet formats it in the RFC 1123 format – the HTTP preferred date format – using a `java.text.SimpleDateFormat`.

RequestDispatcher

When building a Web application, it's common to forward a request to another servlet or to include the output of another servlet in the response. The servlet specification defines the `RequestDispatcher` interface to accomplish this. The support of this feature implies some modifications in the JSPervlet code.

First let's look at why and how the JSPervlet is involved. `RequestDispatcher` lets you forward to another servlet or include the output of a servlet, the forwarded or included servlet being in the same Web application. Since JSPervlet handles all requests toward a Web application, it's invoked.

The first issue is related to the include specification. The included servlet has access to the including servlet's request object. So when JSPervlet is invoked on behalf of an included servlet, the request path doesn't contain its path but the path of the servlet that included it. This is annoying since JSPervlet uses this path to identify the archive and the class to forward the request to. Fortunately it's possible to know the path by which a servlet was invoked thanks to special request attributes described by the Java Servlet Specification, v2.2. For instance, I can get the included servlet `pathInfo` and extract its archive and servlet names with:

```
String pathInfo = (String)request.getAttribute("javax.servlet.include.path_info")
```

If the attribute is not defined, it means the servlet wasn't included, so I can safely retrieve `pathInfo` with `request.getPathInfo()`.

A larger issue is related to the context root. You can get a `RequestDispatcher` with `ServletContext.getRequestDispatcher("/garden/header.html")`. The `"/garden/header.html"` path is relative to the root of the Web application, which doesn't contain the archive name. So JSPervlet won't be able to handle the request. There are two solutions to this problem. The fully standard one is to use relative paths with `ServletRequest.getRequestDispatcher()`. Since we're using `ServletRequest`, the path can be relative to the current request. It addresses the common case where the included servlet is located at the same place as the including one. If it's not the case, you must add the archive name, for instance:

```
ServletContext.getRequestDispatcher(jarName + "/garden/header.html")
```

The problem with this solution is it breaks the independence between development and deployment (where archive names are chosen). In the complete implementation I provide a `JSPHandler.getJAR(ServletRequest)` static method to return the current archive name. You can use it without breaking your servlet portability if you use reflection as shown in Listing 9.

I considered and rejected a fully transparent method. Remember the including servlet is invoked through `JSPServlet`. I could implement a special `ServletContext` that delegates all calls to the `JSPServlet ServletContext` except for `getRequestDispatcher()` in which I'll transparently add the current archive name. I rejected this solution since it forbade invoking a servlet hosted in a different archive. However, if your requirements are different from mine, you can implement this solution.

Now we can come back to the `JSPUpdate` pending issue, to the update handling when the appropriate `JSPHandler` doesn't exist yet. The problem's origin lies in the `JSPUpdate` and `JSPServlet` deployment descriptor (see Listing 1). `JSPUpdate` can't be included in an archive because it would be

unable to download an initial archive, and `JSPHandler` relies on `JSPServlet` init-params to initialize. `JSPUpdate` calls `JSPServlet` when it needs to create a `JSPHandler` and uses a `RequestDispatcher` to achieve this.

Let's revisit the `JSPUpdate` code (see Listing 1). On line 39 you see that when the appropriate `JSPHandler` doesn't exist, `JSPUpdate` creates a `RequestDispatcher` with `getContext().getRequestDispatcher("/JSPServlet")` and uses it to include `JSPServlet`.

`JSPServlet.service()` must be modified to include the code in Listing 10 in order to identify and process updates. This code first retrieves the `JSPServlet` context path using the `javax.servlet.include.context_path` attribute, since the `JSPServlet` is included. Then it invokes `getHandler()`, which will create the appropriate `JSPHandler`. Next, the implementation detects that it's called through a `JSPUpdate` include by checking the included servlet name returned by `request.getServletPath()`. Eventually it retrieves the archive name and remote location from the request and invokes `JSPHandler.update()`, which calls `ClassEntry.update()`.

Beans

`JSP Specification 1.1` provides two mechanisms to integrate Java logic: beans and the more recent tag extensions. A tag extension is arguably more sophisticated and complex. However, it requires no special handling because tag extensions are converted to Java code at JSP compilation time. Therefore we don't need to distribute tag library descriptors and can retrieve tag handler classes from the archive as usual. Though beans are simpler, they can raise an issue in the tool context.

The compiled JSP should create a bean with `Beans.instantiate(getClassLoader(), beanName)`. This static method lets you specify which class loader to use, and its `beanName` can indicate either a serialized object or a class. For example, given a `beanName` of "x.y", `Beans.instantiate` would first try to read a serialized object from the resource "x/y.ser"; if that failed, it would try to load the class "x.y" and create an instance of that class. To fully support beans, I need to allow unserializing from archives.

My code supports the `Beans.instantiate(getClassLoader(), beanName)` way to create the bean (Tomcat code) because target JSPs are loaded by `JSPloader`. Therefore `getClassLoader()` returns the relevant `JSPloader` instance, whose `getResourceAsStream()` is invoked to get the serialized bean.

Summary

It's not difficult to develop a solution that's portable across Java application servers and:

1. Dynamically downloads Web applications from one or many repositories.
2. Commands downloads or updates from anywhere using a browser or a command that can be started from a scheduling tool.
3. Supports Web applications' JSPs and servlets according to `JSP Specification v1.1` and `Java Servlet Specification v2.2`.

A Java server can act as a browser that downloads applets on demand and therefore is administration-free. However, there's a difference. There's nothing to prevent a large number of browsers from downloading the same applet at the same time, collapsing the network. As Java servers download only when commanded, they avoid this problem. ☘

AUTHOR BIO

Alexis Grandemange is an architect and system designer. A Java programmer since 1996 with a background in C++ and COM, his main interest is J2EE with a focus on design, optimization, and performance issues.

agrandemange@amadeus.net

EJB, CORBA, and COM

Maintaining interoperability



WRITTEN BY
SIRL DAVIS

In EJB/CORBA integration, complexity can range from simple to complex and depends in part on the direction of the communication. From EJB to CORBA, communication is relatively simple because the EJB bean invokes CORBA as it does any external resource. CORBA-to-EJB communication, however, depends on the application server's support of RMI-IIOP. If the application server doesn't support RMI-IIOP, then it's best to create a wrapper or adapter class that redirects or delegates the function calls from the client via a CORBA servant, which then calls the EJB.

Part 1: EJB/CORBA Integration

Communicating from EJB to CORBA is the simplest case. The EJB will require an ORB and some means of looking up the remote CORBA object, the servant, in a directory service that could be CosNaming or JNDI over CosNaming. *Note:* It isn't necessary to use CORBA 2.3, which is required by RMI-IIOP (see below). Like any external resource, such as a socket or file descriptor, any ORB can be initialized and used by the EJB.

ORB initialization should be performed once using the singleton design pattern that can act as a factory for looking up remote CORBA servants. A suitable design to ensure this scenario requires the use of a stateless session bean, which is never passivated. Entity or stateful session beans may be passivated and reactivated and would require the disposal and cleanup of the ORB, which has a high performance cost as well.

Using the stateless session bean fits the EJB model best, and the ORB can be a static data member in the bean or in the above mentioned singleton support class to help ensure one instance. A deployment descriptor can also be set, for example, in WebLogic, which allows the container to create only one stateless session bean.

Currently, there's no standard mapping between EJB and CORBA for passing user identity and transaction propagation. So, if transaction and security services are required in your implementation, then the EJB bean wrapping this CORBA call should invoke the appropriate CORBA services in the desired way. In other words there's no automatic solution, and transaction and security will need to be customized for your situation.

In Listing 1 (see www.javadevelopers.com for listings) the CallCorbaBean is a stateless session bean implementation that makes a call to an

object R which initializes the ORB (if it's not initialized) and calls the CORBA servant. The idl is also shown; the servant CORBA implementation isn't included but follows standard CORBA. More important, the R class shows the encapsulation of the CORBA processes in a singleton and as a cascade pattern to ensure single initialization and a simple interface – actually too simple, as an actual implementation would more likely provide factories for different types of objects as required by the application. A client would look up the home interface, CallCorba, and create a CallCorba bean to call the remote function “call.”

CORBA to EJB

Ideally, communication from EJB to CORBA should be over RMI-IIOP. However, RMI-IIOP requires a CORBA 2.3 ORB and an EJB container that supports RMI-IIOP. Because the EJB 1.1 specification only recommends RMI-IIOP but doesn't require it, many of the application servers currently available are in compliance with the EJB1.1 specification and don't fully support RMI-IIOP. In EJB 2.0, released in July 2000, RMI-IIOP is required, and interoperability will eventually be available in all the major container vendors.

Nevertheless, it's still possible to use RMI-IIOP in some application servers, although it's likely to range in complexity from easy to perhaps impossible. For example, Inprise's Application Server product allows easy interoperability with CORBA because it's built on RMI-IIOP as would be expected, as Inprise has integrated it with their Visigenic product. On the other hand, other application servers may not provide RMI-IIOP easily for EJBs and will require creating CORBA idl stubs for all the EJB objects, then successfully compiling the resulting idl with an idl compiler. Sometimes the resulting idl must be tweaked or modified. To create RMI-IIOP between a CORBA client and EJB, follow

these steps:

1. Use `rmic`, which comes with RMI-IIOP, to create idl files from your EJB Home interface.
2. Use a CORBA 2.3-compliant idl compiler to generate your stubs.
3. Create your EJB CORBA client.

As in any new standard, interoperability between vendors is particularly difficult.

Even if the above steps can be accomplished with your application server, there's no standard for passing user identity and the transaction propagation over RMI-IIOP to the client. EJB transaction and security services aren't accessible over RMI-IIOP; consequently, these services will lack any current RMI-IIOP solution.

So although RMI-IIOP is the preferred solution for interoperability between EJBs and CORBA and will certainly be the better solution in the future, it may be required to communicate without RMI-IIOP. One flexible solution is to provide a CORBA wrapper servant that directs CORBA calls to the EJB object. Because this solution can easily be replaced with RMI-IIOP in the future, it will provide a design that evolves.

First, check your CORBA and EJB products. If they easily support RMI-IIOP interoperability, then use that approach. If not, the example code demonstrates a wrapper that delegates CORBA calls to EJB.

In Listing 2, the `call.idl` is the idl for the call to the EJB. The `EJBWrapper` wraps the EJB and allows the CORBA client, `WrapperClient`, to access the EJB through the `EJBWrapper`. The `EJBWrapper` follows the standard adapter or wrapper pattern and simply delegates the calls from the client to the EJB. Note that we're actually using the same session stateless bean used in the previous example for simplicity, although one would never make a call from CORBA

through EJB to CORBA; the example is heuristic.

Design Issues

The design issues regarding communication from CORBA to EJB concern the usage of RMI-IIOP where possible, and where not possible the usage of the adapter design pattern, which ensures that the external resource, the ORB, is carefully handled. The stateless session bean offers the best EJB for EJB-to-CORBA communication because it allows easy management of the ORB. In addition, because the future J2EE software will evolve to support CORBA-EJB interoperability more fully, any effective design must be able to easily absorb future changes.

Part II: COM/EJB Integration

Before looking at COM-EJB communication, it's worth pointing out that SOAP is a strategic direction Microsoft is taking for interoperability. Consequently, SOAP should be considered for communication between EJB and COM. However, it would be a custom solution without the support of current application servers or J2EE technology. In addition, SOAP is relatively new, and not many stable implementations exist.

For some packages using SOAP see www.alphaworks.ibm.com/tech/soap4j or <http://msdn.microsoft.com/xml/default.asp>. For a stable production solution, COM/EJB bridges provide the best synchronous solution and are described below, leaving SOAP for a future investigation.

Bridges

- www.linar.com/: J-Integra is a bidirectional, pure Java-COM bridge.
- www.alphaworks.ibm.com/: Bridge2 Java allows ActiveX objects to be used in Java.
- <http://developer.java.sun.com/developer/earlyAccess/j2eecas/>: Allows COM clients to access EJBs.
- WebLogic's COM Bridge in WebLogic Server: Allows bidirectional COM/RMI interaction.

COM to EJB

Many COM-to-Java bridges exist. Sun has an early access Client Access Services (CAS) COM Bridge available at <http://developer.java.sun.com/developer/earlyAccess/j2eecas/>, that will allow any COM-enabled client to access any EJB object. In addition, bridges such as J-Integra (see www.linar.com/) and WebLogic provide bidirectional communication. Below we look at WebLogic's COM bridge; it's a popular applica-

tion server that simplifies the aspect of bridging between COM and EJB.

Communicating from COM to EJB usually means that Visual Basic clients are accessing EJBs. WebLogic, for example, supports the ability of Visual Basic clients or other Microsoft services to access EJBs. Because bridges support this communication rather well, it won't be addressed in any more detail, and the remaining discussion will be on communicating from EJB to COM, which is more difficult.

EJB to COM

Communication from EJB to COM is necessary if services in Microsoft are required by EJBs. WebLogic's COM solution hosts the COM object as an RMI object. As a representative example, we'll address this bridge in detail.

WebLogic Wraps COM

Note that the WebLogic Server wraps the COM object, Test.dll, and provides an RMI interface any Java client on NT can access. WebLogic provides a "com compiler," which takes a dll file and produces the Java classes, including the interface used by RMI clients. No programming is required, as the code is automatically generated. The command is

```
jview weblogic.comc -nothreads -register
c:\weblogic\examples\com\testser\TestServer.dll
```

Unfortunately, because the standard (Sun) Java Virtual Machine differs from the Microsoft Java Virtual Machine in serialization across RMI, any network connection between the two JVMs is incompatible. Sun Microsystems is suing Microsoft over this and other differences. Consequently, WebLogic certifies that COM will work only if the WebLogic server is run on the Microsoft VM, and only Java clients running under the Microsoft VM will be able to connect to the WebLogic server, so the COM bridge will work only on NT and using only the Microsoft VM. *Note:* Even if the Sun and Microsoft JVMs were interoperable, C++ clients still couldn't connect to WebLogic's RMI. Therefore, the second CORBA wrapper is needed to provide true interoperability (see COM/CORBA/EJB Integration below).

Tests Performed

Three tests of increasing complexity were performed using the above bridge. The first was a simple function that returned void. The second returned a String. The third test returned various Microsoft COM objects that dealt with

the data access (DAO) directly. Both the void and String functions worked successfully, but the more complex test hung after processing one database record and suggests not making Microsoft's Data Access Layer remote. In addition, even if making these COM objects worked easily in Java, the training issue for a Java developer to learn the COM API for data access is knowledge unlikely to be found in one person. Consequently, the best design approach is to return Strings or perhaps XML, simple objects where possible, or domain business objects that encapsulate and hide Microsoft's data access classes. Other bridges may suggest other approaches.

Risks and Disadvantages

One risk to the above approach is the dependence on the COM-Java bridge that's affected by the current lawsuit. Microsoft plans to not support Java in the future and will give Rational the development of the virtual machine (see "Microsoft, Rational strengthen ties on Java development front," www.zd-net.com/eweek/stories/general/0,11011,1015701,00.html). Nevertheless, WebLogic's support of the bridge provides insurance from a major vendor, and there are bridges not bound to the Microsoft VM (see www.linar.com).

Another disadvantage is that some of the features of the WebLogic Server can't be used. Because the Microsoft Java Virtual Machine must be used in order to use COM, clients can't be standard Java clients, which run on standard VMs, although wrapping the EJBs as CORBA objects would allow any client to access them (see COM/CORBA/EJB Integration below). *Note:* JSP technology can't be used because it requires a standard usage of the classpath, that isn't available on the Microsoft Virtual Machine.

Advantages

The WebLogic wrapper provides a simple way to bridge COM and Java with little or no programming. It provides a stable application server, which provides facilities for hosting distributed objects and easing distributed development.

Detailed Implementation Steps

To provide the above solution, WebLogic must run on the Microsoft Java Virtual Machine. In addition, the Microsoft SDK for Java (www.microsoft.com/java/default.htm) must be installed on the machine. After installation, setting its classpath can be tricky, so it's worth noting that this is done in the registry (HKEY_LOCAL_MACHINE\Software\Microsoft\JavaVM\Classpath). Some files can be automatically

added to the classpath and break the WebLogic compiler. To correct the problem if it occurs, simply edit the registry to remove these files from the classpath.

Configure the WebLogic environment file, setEnv.cmd, to use the Microsoft compiler, then:

1. Create a new directory and copy TestServer.dll into it (e.g., C:\weblogic\examples\com\testser)
2. Recompile COM object

```
jview weblogic.comc -nothreads -register
c:\weblogic\examples\com\testser\TestServer.dll
```

3. Copy the created directory weblogic to C:\weblogic\myserver\serverclasses
4. Add the interface, ITestInterface, to jvc /d %CLIENT_CLASSES% ServerSideTestServer.java and write some code to use this interface
5. Restart Web server after adding properties:

```
weblogic.system.startupClass.testserTest=weblogic.com.remote.testserver.ITestInterfaceImpl
weblogic.system.startupArgs.testserTest=TestServer
```

6. Run the above ServerSideTestServer under the Microsoft Virtual Machine, jvc examples.com. ServerSideTestServer

Design Issues

The above can provide a simple way of integrating COM. Especially by using String-level interfaces, XML can be passed as a parameter thereby signifi-

cantly reducing program effort and debugging. Restrictions exist because of the need to use the Microsoft Virtual Machine, making it more difficult to use the full feature set of J2EE. Also, the possibility that SOAP may become a better interface is worth noting. Nevertheless, the need for a production quality solution currently favors a bridge.

Part III: COM/EJB/CORBA Integration

In summary, these solutions can be used together to provide a complete COM-to-EJB-to-CORBA solution useful in those situations where these three object models need to be combined in one solution. Each model provides particular benefits and may be linked to certain legacy solutions in any given environment. Certainly, COM allows easy integration with Microsoft, EJB provides ease of development and portability, and CORBA provides integration benefits. More importantly, in any environment there may be the presence of each model, and there may be a requirement to integrate them all. So, it's worthwhile creating a framework for integrating each model because some loss of performance will be outweighed by development and maintenance benefits.

Performance is a problem with integrating COM and CORBA through EJB. However, although two servers exist between the COM object (e.g., Test.dll) and the Client objects, both components are easily produced as described below. Certainly, this trade-off favors ease of development over performance.

But this solution then provides a bridge between COM and any CORBA client that's highly flexible and can evolve with future changes. Only environments with all three of these technologies will benefit from this integration, but by placing EJB as the central standard, it simplifies the other two integrations. Furthermore, when RMI-IIOP becomes a standard, the server hosting the CORBA wrapper can be removed.

CORBA to COM

To enable CORBA to COM, a CORBA client calls a wrapper or adapter to access an EJB, which then calls COM. Note that RMI-IIOP could be used between CORBA and EJBs, but a more versatile yet performative solution is to utilize the wrapper. The CORBA wrapper delegates any CORBA calls to the RMI interface hosted by WebLogic and provides a fully interoperable solution. The wrapper is essentially a client of WebLogic and looks up the RMI interface of the COM component and provides a remote interface to clients on UNIX or NT. The CORBA interface or idl can be developed manually by creating a matching CORBA interface or by using an automatic tool provided by Visigenic, which can take a Java interface and produce idl (i.e., java2idl). Once the idl is provided the delegation must be coded directly, although this programming is straightforward and could be generated automatically if such a generator were written. Note that by writing a simple code generator that automatically generates the CORBA wrapper, no code would need to be written for the bridge software, which would improve the speed of development and improve quality.

COM to CORBA

COM-to-CORBA communication can be achieved by using the soon-to-be-released CAS, COM to J2EE bridge being developed by Sun, and then having the EJBs call CORBA. Or WebLogic's bridge from COM to EJB can be used and have the EJBs call CORBA. COM to CORBA is in fact simpler than the opposite direction, as would be expected from details discussed above.

Design Issues

COM-to-CORBA communication is enabled by having the COM object presented as an RMI/EJB object by the WebLogic bridge, then a CORBA server is written that wraps the RMI object and provides a CORBA interface to any client. CORBA-to-COM communication is provided easily by having EJBs invoke CORBA directly and by having a bridge between EJB and COM.

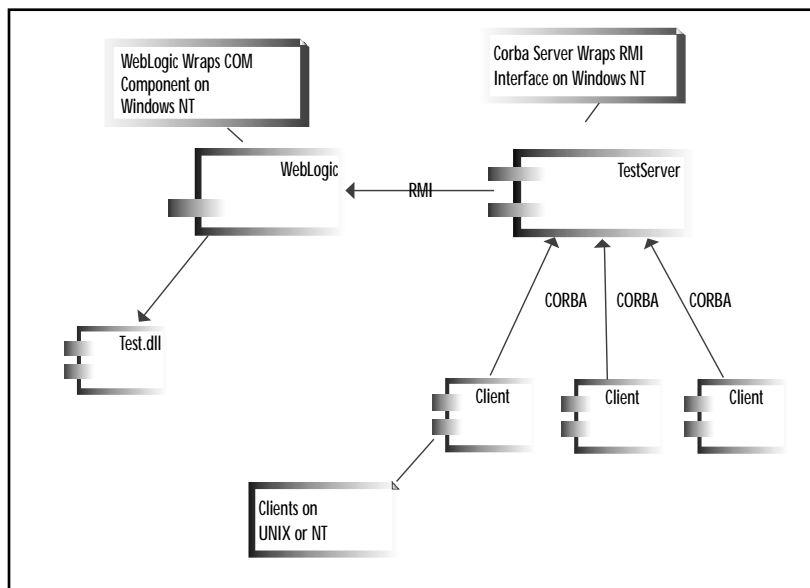


FIGURE 1 Possible COM-to-EJB-to-CORBA integration. Windows NT hosts the WebLogic application server on the Microsoft JVM and the CORBA server, which connects to the WebLogic RMI object.

Macromedia and Java: Serving the Best User Experience

—continued from page 7

The complexity of the interface can be a critical factor in the above approach because a complex interface requires larger amounts of code in the CORBA wrapper. One possibility is to pass XML back from the COM server through the CORBA wrapper, perhaps a hybrid SOAP solution. Automating the creation, the CORBA wrapper can speed development. A second solution is to build common domain business objects and pass them through to WebLogic and Java.

The selection of a particular bridge will be dependent on platforms and overall configuration. If WebLogic's bridge is used, then the Microsoft VM will be required, as it runs only on Windows NT. Other bridges will have different requirements. However, an EJB container does simplify many of the interoperability issues by providing one standard to map both CORBA and COM rather than mapping each standard to the other, which would create six mappings: CORBA to COM, CORBA to J2EE, and so on.

AUTHOR BIO

Sirl Davis is an assistant director of IT at Dresdner Kleinwort Benson, London.

He has a bachelor's degree in electrical engineering/computer science and an MBA/finance with PhD-level work in information systems. His experience includes developing various finance applications for trading, risk, and data warehousing utilizing Java technology.

Conclusion

EJB, CORBA, and COM are different technologies and are likely to persist in production environments. Maintaining interoperability is essential for environments that have these technologies. A number of issues have been discussed that show many of the choices required for integrating these technologies. Various trade-offs and advantages exist for different approaches and tools, and any particular environment or configuration of software will favor a different combination of these solutions. ☛

sirl@mail.com

graphics from template files, data sources, and external graphic resources.

A portion of Generator server is written in Java. When you install Generator, the Java Runtime Engine (JRE) and Allaire JRun are also installed to provide Generator with Java components to execute. Generator uses Java for two primary purposes: initiating generation and acquiring data from external sources.

When initiating Generator, a Java Servlet engine allows Generator to pass information to and from the Web server via the Java Servlet protocol. Java and Java Servlets are used to trigger Generator from a Web server or from the offline Generator application. Macromedia chose Java servlets for their high performance, reliability, and flexibility. Like other Web server APIs, such as CGI and Netscape Server API (NSAPI), the Java Servlet API from JavaSoft offers a way to extend the functionality of Web servers.

Generator offers flexibility to your Web application when accessing Java data sources. Data can be directly accessed through a result set from a SQL query passed through JDBC/ODBC, Java class files, or other formats.

Macromedia partners with leading technology companies that support Java to ensure that its authoring tools integrate with the latest technologies and third-party solutions. In addition to its multiyear partnerships with Sun and Allaire, Macromedia also has close relationships with ATG, BEA, BroadVision, IBM, and Vignette to provide development tool support for their Java solutions.

Macromedia is committed to the Java platform to ensure that its developers can deliver the most effective and engaging Web content possible using their technologies of choice, and can use Java wherever it's appropriate as they continue to define what the Web can be. ☛

Modeling Enterprise Java Components with UML



WRITTEN BY
VAUGHN VERNON

It's a pleasure to be writing this article in the beautiful, high-tech, Mediterranean city of Netanya, Israel. It's the evening after the first day of the workweek here – Sunday, of course – in my current consulting engagement. But working Sundays isn't the only thing I've had to get used to here.

Although I've driven a car in at least 12 different countries on several continents, I've never had a greater potential for road-way dyslexia than I have here. Not only is the Hebrew language foreign to my ears, the script found on street signs is written right-to-left (at least that's what I've been told) and doesn't even remotely resemble the Roman-based characters I've grown quite fond of. Oh, and the Hebrew text is generally followed by a second line of text in Arabic. Hebrew with Arabic – my guess is that it's the highway department's answer to 128-bit encryption.

What do my language deficiencies have to do with representing Enterprise JavaBeans in the Unified Modeling Language (UML)? Everyone needs a common method to communicate thoughts and ideas. In Israel the international language is, thankfully, English, which in many cases is the third line of text hosted on road signs here.

But it doesn't end there. As I present Java 2 Enterprise Edition course instruction and discuss project strategies for migrating existing proprietary application server components to the J2EE architecture, I speak freely in English as those who speak native Hebrew, Russian, and French listen and participate. It works because we have English in common. But when it comes to communicating software architectures and designs in general, and EJB specifically, why do so many often resort to "foreign" methodologies and syntaxes or even choose to "grunt" instead?

Why Not UML?

From my vantage point, there's been an explosion in the use of UML. When we

all come from differing software development backgrounds, we can make architecture and design work for us because we have UML in common. But why do some hesitate or even fail with UML?

Frankly, I believe that many start off with good intentions to learn and use UML. But because of work environments that lack a reliable software process, capable engineers end up abandoning a standards-based modeling approach for the "code by the seat of your pants" antimethodology. Generally I've found this is due to a lack of understanding and/or commitment at the top. Because they've never used or understood a software process themselves, many managers are all too willing to let their development staff live with the pain. So for those of you who think you need to be good looking to be hired for an "Object Modeler" position and that "Senior" is completely out of context, you'd better visit your favorite online bookstore and order a copy of Martin Fowler's *UML Distilled* (Addison-Wesley) immediately following your plastic surgery.

On the other hand, many of you are completely sold on using a software development process with UML, but may find the cost prohibitive. True, the cost of UML tools can be out of the stratosphere, but there are options. At least one of the leading UML vendors has expressed willingness to configure its product to fit your budget by unplugging costly features such as round-trip engineering. Granted, round-trip engineering may sound appealing, but one UML authority told me it shouldn't be used by organizations with a Capability

Finding common ground

Maturity Model that's less than Level 4 or 5 (www.sei.cmu.edu/cmm/).

Trying to maintain a set of object architecture and design artifacts that are perpetually synchronized with the source code of a complex software system can be a daunting task (with one exception explained below). So why pay for a feature you may find too difficult to use? And if you're just looking for a way to try out the various UML diagrams, there's at least one pure Java Open Source UML tool, called Argo/UML, available free of charge at <http://argouml.tigris.org/>. Argo/UML has been somewhat limited in the past, but has many more diagram types in the latest version.

Whatever your budget, note that UML is an understandable and generally concise syntax. While there are a number of diagram types – officially nine in all – most of us will make good use of about four or five. I recommend mastering at least use case, collaboration, class, sequence, and a combination of package and component diagrams. Also useful for some stages of design are state and activity diagrams. You can take a phased approach to UML, perhaps starting with use case and class diagrams, then adding a new diagram type as you see fit. Also be aware that the software development process you follow will greatly influence which diagram types you use and at what stage in a development iteration you use them (see sidebar, "When to Use UML Diagrams").

But there are issues for those of us who know UML and want to make it work for our middle-tier, enterprise-component development, namely Enterprise JavaBeans.

Modeling Enterprise JavaBeans in UML

Unfortunately, the UML doesn't at present support the expression of every software idiom we need to use. While there's excellent support for the Java language in general, and simple JavaBeans components specifically, there's currently no standard definition of how to model Enterprise JavaBeans in UML. Granted, there are ways to model EJB in UML, and as you'll see, depending on the tool you use, it might work out very well. But the Enterprise JavaBeans architecture was specified well after the UML was completed. Because it's a complex specification, there proved to be aspects of the Enterprise JavaBeans component architecture that the designers of the UML didn't previously consider. What that spells for you and me is the potential for UML modeling tools to be difficult to use when designing EJBs. However, UML designers exercised appropriate forethought in these situations and built extensibility features into the language so it can grow into the tool it needs to be for any given software model.

How will the UML be extended to support Enterprise JavaBeans development? The Enterprise JavaBeans Specification version 1.1 stated under Appendix A entitled *Features* deferred to future releases: "We plan to enhance the support for Entities in the next major release (EJB 2.0). We're looking into the area of use of the UML for the design and analysis of Enterprise Beans applications."

Unfortunately, the proposed final draft of the version 2.0 specification doesn't even mention the UML. If previ-

ous performance is any indication of the future, we can't expect to see the next specification for another year or so. However, we may get an answer sooner than that.

Java Specification Request Number 26

It appears that the EJB specification group has turned over the UML/EJB specification to the Java Community Process Program, who has assigned it a specific Java Specification Request number, JSR-026. I must emphasize the words "it appears" because it's actually difficult to understand what's happening with this whole subject. I've attempted to contact a few members of the JSR-026 Expert Group, including the specification lead, but have failed to be granted any insight into the progress and status of the work accomplished thus far. However, I can tell you which Enterprise JavaBeans development issues are being discussed.

In general the UML/EJB specification will focus on creating extensions to the UML that will allow tool and framework vendors to provide standards-based modeling capabilities and source code synchronization facilities between the UML and EJB implementations. While the common round-trip engineering features of many UML tools focus on model-to-source-code synchronization, a giant leap forward will be the definition of standards for storing UML models in the EJB-JAR file and associating the model components with the actual software components they represent.

Companies creating reusable EJB components will then be able to bundle UML models of their components to permit tool and framework vendors to use their models to automate the consumer's use of the components. Perhaps some component development organizations are already shipping UML models to their customers, but likely such models are supported by only a single UML tool vendor. The real strength behind standardizing an EJB-JAR UML model archive is to allow all models to be read by any number of standards-based tools and frameworks. Thus, the UML models become just as open and reusable as the Enterprise JavaBeans components they represent.

So we're really talking about empowering both EJB component vendors and EJB component consumers, which will become increasingly separate camps over the next few years. EJB component vendors will have the tools they need to rapidly develop and deploy server-side components using UML-based tools that understand the logical and physical EJB constructs to UML model elements, including sophisticated and vastly improved round-trip engineering facilities. Once the EJB components and UML model bundles are delivered to the component consumer/customer, he or she is empowered to quickly examine the server-side object model diagrams and use them to define relationships between their newly acquired EJBs and their existing software. Hopefully with a few drags and drops and some relatively simple glue code, the new components will soon be in live production!

When to Use UML Diagrams

While anyone can learn to draw pictures of software, it's sometimes *when* you draw them that matters most. One failure of many books about UML is that they spend a lot of time dissecting the syntax into excruciating detail, but they don't really teach you when the various diagrams should be used. Be aware that the software development process you follow will greatly influence which diagram types you use, and at what stage in an iteration you use them. Therefore, I can't provide absolute direction, only some general guidelines based on my own and others' experience.

Many teams are finding success using an iterative software development process. By iterative I mean that relatively small chunks of software are created in relatively brief development cycles. Once a small software deliverable is defined, a cycle called an iteration or "timebox" is begun. Upon successful completion of the iteration, a release is performed. Then another small software deliverable that may complement a previous iteration is performed. After each iteration is integrated into the release, software testing can begin on the latest deliverable. At some point enough iterations will be complete and stable enough to constitute a product ready for alpha and beta test, and eventually final release.

• Step 1:

Use Case Diagrams: At the beginning of the product conception stage, the nontechnical and technical teams will merge to perform some high-level requirements analysis and create a product vision document. Following full consensus, the same team will work together to create perhaps 10–12 high-level use cases that clearly define the product's intended use. At this point both text documents and UML use case diagrams can be used to fully express the product's requirements. Then, the first development iteration begins. The test team will use the use cases to begin developing a test plan, while at the same time the development team will use the use cases to create lower-level use cases for defining software requirements. The resulting software requirements can then be shared with the test team to help flesh out their test plan into test requirements. Many times this work will be performed in unison.

• Step 2:

Collaboration Diagrams: Following the use case portion of the iteration, collaboration diagrams can be created to look at the behavior among the various objects in a single use case (one bubble, most likely including any directly accessing actor). At this point things are becoming more concrete. We're transitioning from relatively high-level models to the discovery of which objects will be alive inside a use case, and what method invocations cause the collaborations.

—continued on page 42

One of the JSR-026 Expert Group deliverables is an XML DTD (Document Type Definition) describing the valid format for the UML models housed in the EJB-JAR file and the relationship of model elements to EJBs in the same deployment file. The UML model file stored in the EJB-JAR will be XMI, an XML-based definition designed to facilitate the storage of UML models in an open, vendor-neutral format.

It appears that JSR-026 won't be fully incorporated into the EJB specification until, perhaps, version 3.0. Since JSR-026 deals with issues of deployment, the UML/EJB specification may not be provided until there's some resolution to standardizing deployment of a single EJB-JAR file to any number of compliant servers. On the other hand, the Expert Group may release a specification before you've read this article. You can check on the progress of the specification by navigating to this Sun URL: http://java.sun.com/aboutJava/communityprocess/jsr/jsr_026_uml.html. Either way, we need to model server-side components. So what can be done at the present time?

Together Control Center

The JSR-026 home page states concern over vendors who will use proprietary scaffolding to host EJB modeling in the UML tools. In the long run we'll be happy if all UML tool vendors implement their EJB wiring according to specification. But in the meantime I'm more than

happy to make use of whatever's available to help me get my work done. And Together Control Center from TogetherSoft is one of those tools. Together Control Center is a top-of-the-line product for Java-based modeling and does an excellent job of supporting the creation and deployment of Enterprise JavaBeans to a variety of servers. From the quality of their proprietary EJB modeling tools, I believe that TogetherSoft will have little trouble moving to a specification-conforming edition, that is, once the JSR-026 Expert Group catches up to them.

If you've ever developed an Enterprise JavaBean, you know there are a lot of parts to keep track of. If you're creating a session bean, you have three Java source files to maintain – the remote and home interfaces and the bean's implementation class. If you're creating an entity bean, you'll have to add another file to support your entity's primary key if the primary key is too complex to be represented by a Java String or a java.lang wrapper class (e.g., Integer). Just making sure all your remote and home interface methods are properly implemented in your bean class can be challenging. Add to that the creation of the `ejb-jar.xml` deployment descriptor as well as server-specific deployment descriptors for resource management and object-relational database mapping, and you realize there's plenty of room for error.

Together Control Center helps simplify EJB development. While you still have to create all the files you did before,

Together Control Center minimizes the number of errors you can produce. Most useful is the UML class diagramming for EJBs. Two toolbar buttons support EJBs – one for session beans and one for entity beans. Let's take, for example, the *Session EJB* button. If you click the button and then click the diagram, a UML class element representing your EJB session bean's implementation class appears. Automatically generated for you is the bean's `SessionContext` instance variable, as well as the `setSessionContext()`, `ejbActivate()`, `ejbPassivate()`, and `ejbRemove()` methods, and a default `ejbCreate()` method.

As you add elements to the class, such as new business methods, you may not realize what's being done for you in the background. If a newly added element has a corresponding element that must be synchronized with the remote or home interface, Together Control Center automatically adds the element to the proper interface. For example, if I add a business method named `getUsersByAccountId()`, the same method definition will be added to the bean's remote interface. Adding a new `ejbCreate()` method, say, with an initialization parameter or two, adds a corresponding `create()` method definition to the bean's home interface.

By default you don't see the EJB's home and remote interfaces, only the implementation class. That's great when you're first designing your component. By hiding the remote and home interfaces you save a lot of diagram real estate, and

—continued from page 40

- **Step 3:**
Class Diagrams: While collaboration diagrams introduce objects, they generally indicate very little about static relationships between the collaborating objects. Static relationships defined in class diagrams include object types and super types, special interface implementations, navigation and dependencies, how the objects are composed and aggregated, and what additional properties and methods must be created to support the real world. At this point most developers are itching to cut the code. But if you're just a little more patient it will pay off.
- **Step 4:**
Sequence Diagrams: Class diagrams are among the static diagrams because they don't indicate behavior, only compositional relationships. Collaboration diagrams are good for discovering new objects, but they're not so good at documenting collaboration sequence. In fact, sequence can get a little hairy in them. But sequence diagrams are great for defining the interaction between objects of various types at the method invocation level. By providing sequence diagrams of your object interactions you'll be able to analyze your class designs before the code is cut, and provide clear direction to those who actually write the code.
- **Step 5:**
Package and Component Diagrams: Now you need a place to house the classes you've defined. Package and component diagrams are used to define object dependencies and relationships at a much higher level

than class diagrams. Actually there's really no such thing as a package diagram, per se. Rather, you can "invent" package diagrams by inserting package icons with interpackage relationships and nested packages on a static structure diagram (class). Package diagrams can map one-to-one into Java class packages. A single component object in a component diagram can map one-to-one to a single Enterprise JavaBean and define its navigable dependency on other middle-tier components. Use these diagrams to help your team understand how the current iteration will be packaged and how it can be snapped together and deployed with existing software from previous iterations.

If you hand over a set of well-defined collaboration, class, sequence, package, and component diagrams to an experienced group of programmers, they'll have little if any question of how the iteration must be coded and deployed.

But what about the software architect's role? Actually the architect's role will begin in Step 1. A single architect or group of architects could be called on soon after the product requirements are completed, since at that time the target platform(s) have been defined. A system architecture will be created even before the first development iteration begins. The resulting architecture document won't be limited to a single type of UML diagram since it's impossible to express a complete architecture using a single kind of modeling diagram. While it's out of the scope of this sidebar text to explain how to provide a complete architecture document, you should consider reading about the 4+1 View Model approach to software architecture on www.rational.com/products/whitepapers/350.jsp.

you're less distracted by the additional pieces. However, when I start to define my client-side classes, such as one supporting the Verge EJB Client Design Pattern (a

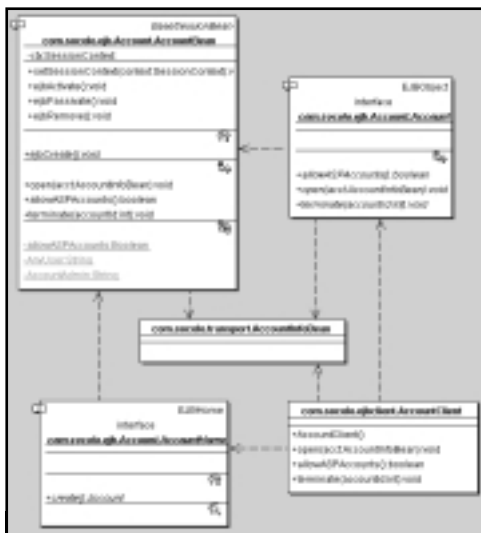


FIGURE 1 A UML class diagram of a simple account session bean using Together Control Center

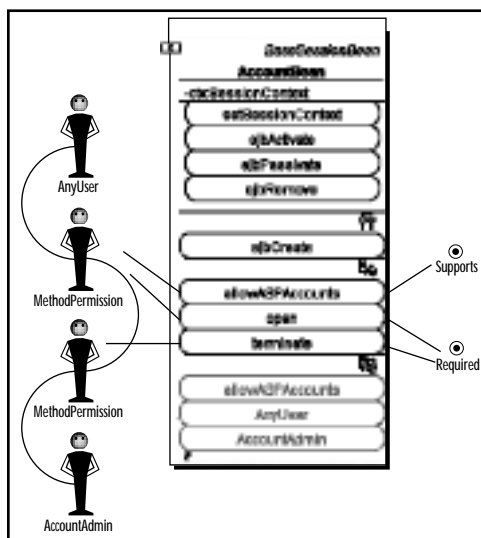


FIGURE 2 An EJB assembly diagram specific to Together Control Center

AUTHOR BIO

Vaughn Vernon is a principal Enterprise Java architect with Verge Technologies Group, Inc. (www.vergecorp.com). He has 18 years of experience in software architecture, design, and development with a focus on the use of industry-standard, leading-edge, object-oriented technologies, including J2EE and UML. Verge is a Boulder, Colorado-based firm specializing in e-business solutions with Enterprise JavaBeans.

wrapper class that hides the details of the J2EE/EJB APIs and other extras necessary for client interaction), I prefer to model the client by showing navigation dependencies to the remote and home interfaces as opposed to showing the client conversing directly with the EJB implementation class (which in fact it can't do).

To make the remote and home interfaces visible, you simply use the View/Diagram View Management... properties sheet to toggle the EJB view settings. Figure 1 shows the results of just a few minutes of EJB modeling with Together Control Center.

In addition to adding a new business method or ejbCreate() method, you can add the standard EJB deployment refer-

ences for environment, bean, security role, and resources. In Figure 1 I've added a special bean environment entry named "allowASPAccounts" as a Boolean. If you're setting up special transaction attributes and security roles, you may want to make use of the special EJB Assembly Diagram provided by TogetherSoft as a proprietary UML extension. As you can see in Figure 2, the assembly diagram states that "Any-User" can open() an account and query for allowASPAccounts() support, but you must be an "AccountAdmin" to terminate() an account. It also sets the required transaction attribute for the open() and terminate() methods, but states that allowASPAccounts() supports transactions but never initiates one.

Time and space don't permit me to delve much deeper into Together Control Center, but one last feature I must mention, which by the way isn't specific to EJB, is the unique round-trip engineering capabilities of Together's product line. There's no need to synchronize your source code to your model, or your model to your source code, because your Java source code is your model's archive. When you create a new class in Together Control Center, it generates a Java source file automatically. If you then vastly alter this Java source file using a programmer editor, save your changes, and return to Together Control Center, your changes are immediately reflected in your model. In fact, a major strength of Together's products is that you can actually use them as your development environment. TogetherSoft calls this unique feature OneSource Simultaneous Round-Trip Engineering.

You can try out Together Whiteboard edition for free without any size or time limits by downloading it from <http://togethersoft.com>. The Whiteboard edition is limited to only class diagrams, but includes all the basic IDE features of the full product and GoF (Gang of Four) and other design pattern plug-ins.

Deploy on JBoss/Server

In conclusion I'd like to shift gears a little and tell you about one of my favorite Java 2 Enterprise Edition subjects of late. It's the JBoss Open Source Enterprise JavaBeans Server project (www.jboss.org). I started using JBoss last July and found it to be a fascinatingly wonderful product! The contributors to the JBoss project have produced a truly first-class EJB server and container. Last summer as I evaluated the code drops that eventually led to the recent JBoss version 2.0 product suite release, I thought I'd stumbled onto something

worthwhile. Having been described as "coders on steroids," the JBoss team, led by Marc Fleury and Rickard Öberg, has produced an incredible amount of solid code in a relatively short period of time. But they're not just coding. The developers do such a good job of managing the support forums that they've attracted an additional bunch of helpful contributors to the project. Personally, I'd like to thank Darius Davidavicius of Lithuania and Torsten Terp of Denmark for patiently helping me create an Oracle data source for JBoss/Server.

As you're probably guessing, all is not perfect. I didn't struggle with the data source creation for no reason. There are some serious holes in the bowels of the JBoss documentation. And while there are already EJB 2.0 features cropping up (such as a recent preliminary implementation of message-driven beans), how long can a group like this go on without some nontechnical assistance? A large-scale project like this will eventually begin to have some large-scale needs. In the spirit of the Linux and Apache projects, I believe that a project of JBoss' magnitude is a natural fit for large-scale financial support.

Recently the aforementioned Darius reported that JBoss/Server is outperforming the market's leading EJB server by over 20% as both servers run his deployed code side-by-side. When you consider that non-e-commerce, three-tier applications are beginning to be sold into business enterprises on CD-ROMs with EJB servers as the infrastructure, JBoss/Server has removed the per-CPU licensing barriers to those markets. Our Israeli client is a testimony to that, and they're using JBoss for that very reason. Add to that the array of business-to-enterprise start-ups coming down the pike and the millions of dollars that can be saved on developer and deployment licenses. What could that be worth?

Of course, there's always going to be a place for the market-leading, top-priced EJB application servers. It's a given, the same given that some variety of Windows is still outselling Linux by a long shot. But just as there's a place for Linux, there's a place for JBoss. In the short term I'd at least like to see TogetherSoft create an EJB deployment interface for JBoss.

At a minimum, I recommend you download the JBoss product suite if you haven't already done so. See if you don't agree that it's one of the tightest, well-architected pieces of software you've used in a long time. If so, think about contributing in some way to the effort, either technically or otherwise. ☎

vernon@vergecorp.com

Journeyman's HTTP Driver



WRITTEN BY
MARC CONNOLLY

If you were to consider all the surface clutter in my life, you might find it strange to know, deep down inside, I like things simple. I mean really simple, because I'm not very bright and have trouble with basic concepts, like getting out of bed and getting the car pointed in the right direction on Monday mornings.

I'm also cheap. Folks who know me will tell you they've seen me pick up pennies from the street, but this is simply untrue. I stoop for nickels and dimes when people are nearby. Pennies I always walk past and fetch afterwards when no one is looking. By the way, this is a secret I'd like kept between you and me, okay?

There's something else. The past 10 years of travel have left me feeling a bit like a journeyman craftsman, not too unlike a coppersmith of old who traveled from town to town plying his trade and wares. Given this trait, I tend to gravitate toward things that are lightweight and portable – the more, the better.

These particular attributes came to the surface recently when I had to perform a volume test against a servlet-based application I'd written. I wanted to get a feeling for the behavior of several key pooling components and how they fared under stress without going through a lot of trouble or expense.

So I went looking for an existing solution in that great ether we call the Internet and which I have occasionally, in a less charitable frame of mind, referred to as *the great flotsam*.

HTTP Driver Criteria

My criteria were simple. What I needed had to meet three basic requirements. First, because I'm a simpleton, the solution had to be easy to set up and use.

Second, it had to be economical because, remember, I'm cheap and didn't want to spend a lot of time or money.

Finally, I preferred a solution that

was reasonably portable. I didn't want something I would have to make or go through a lot of trouble setting up and reconfiguring every time I moved to a different environment.

So with those requirements in mind, I spent some time looking around for anything that appeared promising; I felt pretty sure, within an hour or so, that I'd run across an acceptable solution. However, and a bit to my annoyance, I came up empty-handed.

Well, that's not exactly true, because I did find other folks who were interested in the same thing, and I found references to commercial products, a few of which I'd worked with in the past.

But given my criteria (simple, cheap, portable), I decided I'd looked around enough. So, sitting back in a huff, I considered my options:

1. Give the app to the end users as is and hope it works
2. Hire and train some chimpanzees to do the volume tests
3. Quit, move to another country, and hope no one comes looking for me

Option 1 had seldom worked in the past and, given the number of times I'd tried this approach, odds weren't good that it would be any different this time. On the surface, option 2 seemed to have some merit until I recalled the energy expended training another primate more closely related to me, namely, my son. Option 3 was out of the question because they always come looking for you, wherever you run.

Deciding I was out of alternatives and the need to do a volume test wasn't going away, I decided to take a more pragmatic approach and try to come up

with something on my own.

After a bit of doodling I decided that with a little effort, a viable solution was possible in Java that would meet two of my requirements. First, developing the solution in Java would give me the portability I was looking for. Second, the richness of Java would provide me a level of abstraction that would remove me from the grittier aspects of, say, socket programming details. Java's richness would also lend itself to the economies of time and expense I was looking for. However, the "simple, easy to use" requirement would be up to me.

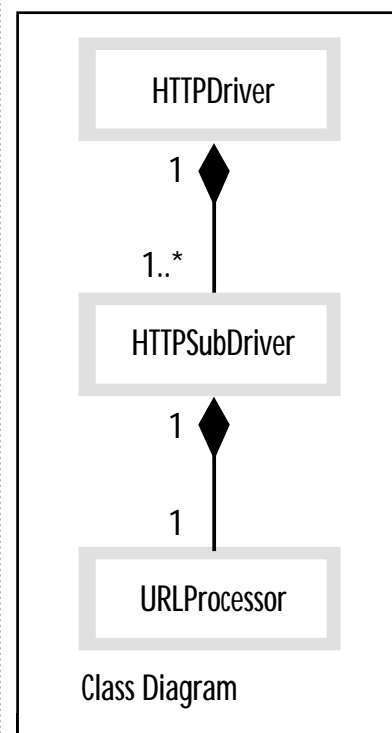


FIGURE 1 UML diagram

HTTP Driver Solution

The end result was a trivial Java-based application employing sockets and threads to issue multiple HTTP requests – and process responses – repetitively, against one or more HTTP servers. Roughly speaking, the application can simulate the activity of many users.

The structure of the application is quite simple (see Figure 1).

The application is composed of three classes – HTTPDriver, HTTPSubDriver (which extends Thread), and URLProcessor. HTTPDriver creates one or more instances of HTTPSubDriver, each of which, in turn, creates a single instance of URLProcessor. An instance of HTTPSubDriver is analogous to a single user, while the URLProcessor could be thought of as the repetitive serial actions (“click stream,” if you prefer) taken by that user.

Initially, HTTPDriver’s main purpose is to read at least two parameters. The first indicates how long HTTPDriver is to run. The second specifies the location of another file that contains information on the number of “users” (HTTPSubDriver instances) to create and the runtime details for each.

For example, HTTPDriver could be invoked as follows:

```
java HTTPDriver 20 subDriverDefs.txt
```

The first parameter, 20, indicates the maximum amount of time the entire process is to run. The invocation of HTTPDriver would run for a maximum

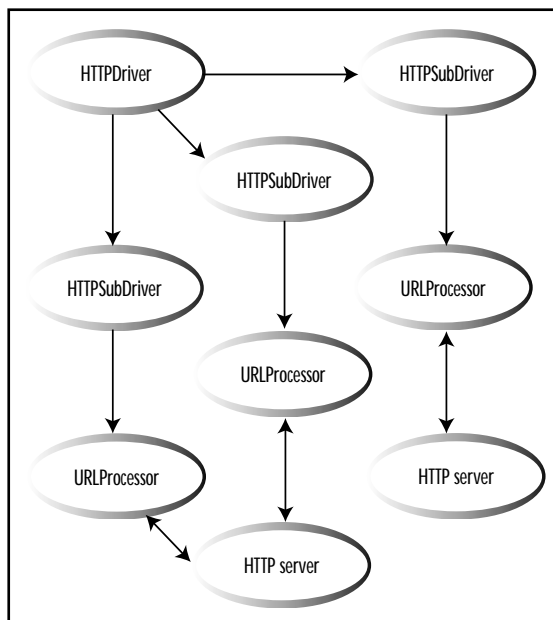


FIGURE 2 One HTTPDriver instance with three HTTPSubDriver instances

of 20 minutes. The second parameter is the name of a file that contains a list of “users” definitions or, more precisely, HTTPSubDriver definitions.

HTTPDriver always produces a log that, by default, is called HTTPDriver.log and is created in the current directory. You could, however, override that by supplying an alternative log file. For instance:

```
java HTTPDriver 20 subDriverDefs.txt
\tmp\HTTPDriver.log
```

Again, each user is represented by an instance of HTTPSubDriver with its own set of parameters. Using the preceding command line invocation as an example, the parameters for the HTTPSubDriver instances would have been in the file called subDriverDefs.txt.

These parameters convey, for example, the subDriver’s own individual runtime (which could be less than HTTPDriver itself), its sleep time, the file containing the list of URLs to be processed, and the maximum number of passes URLProcessor is permitted to make upon that file within the time HTTPSubDriver has been allotted to run.

Using the preceding command line invocation as an example, subDriverDefs.txt could contain something like this:

```
TEST_1 1 10 www.dev.com:80
EchoServlet.txt 3 2000
TEST_2 10 15 www.acp.com:80 empSe-
lect.txt 9 4000
DML_1 5 3 www.dev.com:80
EmpDML1.txt 8 2000
DML_2 1 10 www.dev.com:80
EmpDML2.txt 7 2000
```

Each line represents the parameters for a single instance of HTTPSubDriver. The first parameter is an arbitrary tag that HTTPSubDriver uses whenever it writes to the log. The second parameter is the maximum runtime, in minutes, the HTTPSubDriver instance is to execute, while the third parameter specifies the HTTPSubDriver’s sleep interval in seconds.

As you can probably deduce, the fourth parameter is the host and port of the HTTP server against which the requests will be directed and the responses read from. Note in the example I have www.dev.com and www.acp.com specified, presumably referencing two different hosts. There’s nothing to stop you from driving traffic to different hosts within the same HTTPDriver instance.

The fifth parameter is the name of a file containing the list of one or more URLs (more on these later) to process.

The sixth parameter indicates the maximum number of passes the URLProcessor can make on that file. The seventh and final parameter is the size of a character buffer into which segments of the response stream are read.

Now that was a mouthful but, again, the implementation is quite simple: HTTPDriver creates one or more HTTPSubDrivers, each of which creates a URLProcessor to do the work (see Figure 2).

Internal Structure

Let’s look at some of the code at a high level. As part of its initialization, HTTPDriver creates a Vector on its HTTPSubDriver instances to monitor and control the instances collectively:

```
private Vector threadPool =
    new Vector();
```

As each HTTPSubDriver instance is created, its handle is added to the Vector:

```
threadPool.add
(new HTTPSubDriver(
    subDriverId
    ,subDriverRuntime
    ,subDriverSleepTime
    ,hostAndPort
    ,subDriverInputFile
    ,maxInputFileIterations
    ,httpResponseBufferSize
    ,logTimestampFormat
));
```

During its instantiation, HTTPSubDriver creates an instance of URLProcessor:

```
urlProcessor =
    new URLProcessor
    (subDriverId
    ,hostAndPort
    ,subDriverInputFile
    ,maxInputFileIterations
    ,httpResponseBufferSize
    ,logTimestampFormat);
```

After HTTPDriver has created all the HTTPSubDriver instances, it marks them as daemons and starts them:

```
for (int i=0;
    i<threadPool.size();
    i++)
{
    HTTPSubDriver httpSubDriver =
        (HTTPSubDriver)
        threadPool.elementAt(i);
    httpSubDriver.setDaemon(true);
    httpSubDriver.start();
}
```

Remember, as each `HTTPSubDriver` was created, an instance of `URLProcessor` was also created. Generally speaking, `URLProcessor` reads the file it was given that contains a list of URLs. For each URL encountered, `URLProcessor` opens a socket to the server, writes the request, waits for and then reads the response, and finally closes the socket. `URLProcessor` does this one request after another. The requests can be any valid HTTP method but were typically GET and POST in my case. There's nothing to stop you from including other methods, such as HEAD, DELETE, OPTIONS, or so forth, in the list of URLs. PUT would require a few extra lines of code, however, similar to the way FORM data handling is described below.

`URLProcessor` recognizes two formats:

```
{Method} {URL} HTTP-version
```

or

```
{Method} {URL} HTTP-version FORMTA={}
```

where FORMDATA has the following format:

```
keyWord1=value1&keyWord2=value2...
```

A typical file containing a list of URLs

might contain many entries and will look something like this:

```
GET /dev/servlets/Login HTTP/1.0
GET /TFGreenOnBlack.jpg HTTP/1.0
GET /stylesheet.css HTTP/1.0
GET /applets/MenuManager.jar HTTP/1.0
POST /dev/servlets/Login HTTP/1.0
FORMDATA=userId=Frank&userPswd=Zappa
```

Note the FORMDATA literal in the last sample line. This is a keyword that `URLProcessor` looks for to determine whether there's any FORM data to be appended to the request as its entity body.

With the exception of the last line, this sample looks very much like a typical HTTP server's access log. Also note that if you wanted to gather the "click stream" of a particular application, the HTTP server's access log would be an ideal source to begin with. Or you could do something similar to what I did. Within my application I embedded two fragments in the `doGet` and `doPost` methods, respectively, which enabled me to quickly generate "click streams" for playing back through `HTTPDriver`. I've since folded those fragments into an existing general utility class reducing the code to a single line in each method.

`URLProcessor`'s `next()` method is where all the activity happens and

where the requests to the HTTP server are issued and the responses handled. `URLProcessor` is responsible for reading each URL from the list, forming the request, creating a socket to write the request stream to, and subsequently reading the response stream from. After each request is written and its response completely read, the socket is closed – indirectly by a `.close()` on the response stream – and a relative response time duration is calculated and logged.

This repetitive process is governed by dispatching each instance of `HTTPSubDriver`. As each instance is dispatched, it invokes its `URLProcessor`'s `next()` method and continues to do so until interrupted.

As it runs, `HTTPDriver` periodically examines the state of all the `HTTPSubDrivers` it created:

```
while(System.currentTimeMillis()
    < driverStopTime)
{
    logMessage
        ("Current Elapsed Runtime: "
         +
          (System.currentTimeMillis()
            - driverStartTime)
           / 1000 + " seconds");
    if (backgroundThreadsActive())
    {
        Thread.sleep(10000);// 10s
```

```

    }
  else
  {
    logMessage
      ("All background threads
       are inactive");
    break;
  }
}

```

If all of the HTTPSubDriver instances are no longer active, HTTPDriver closes its log file and terminates. That's it. As you can see by this description, the entire process is fairly straightforward.

Runtime Considerations

There are a few things you should consider.

Remember when I said that this solution can simulate, roughly speaking, the activity of many users? Well, I qualified the statement for a reason. If you think about the nature of this application's structure and implementation, you'll recognize some limitations. For instance, we're trying to approximate the behavior of multiple users, but what happens if we run a single instance of HTTPDriver with 20 HTTPSubDriver instances – that is, 20 users?

First off, the runnable queue could get quite long, potentially elongating the

duration between dispatch for some threads. This problem could be pronounced if your environment is not conducive to parallelism – for example, running in a single CPU configuration or using green threads, which is the default behavior. The implications of this problem are such that some of the HTTPSubDriver instances might not get dispatched frequently or long enough to do anything substantive.

Compounding the issue of a long runnable queue could be the interwoven effects of normal thread blocking, for instance, on socket read waits. Making things further problematic, I've set each HTTPSubDriver thread to Thread.NORM_PRIORITY and haven't considered systems where time slicing may not be supported. For example, I didn't implement yield() in the construction of run() in HTTPSubDriver.

Something else to think about would be network congestion along the routes taken by this application. But more important would be the congestion at the point of origin – in other words, where HTTPDriver is running. For instance, in my tests, when I ran a single instance of HTTPDriver from one machine using 10 HTTPSubDriver definitions, the behavior was adequate. However, when I started multiple instances – three in mv case – of HTTP-

Driver on the same machine, each with 10 HTTPSubDriver definitions, things got a bit slow. Aside from the fact that I was running on a single CPU, I was choking my NIC.

By reworking the test configuration (two HTTPDrivers, each with six HTTPSubDriver instances per machine) across several “client” machines, I got a reasonable amount of traffic resembling the transit/response times of a single instance of HTTPDriver on a single machine.

These observations about runnable queue length and network congestion yield a key practical consideration in the application of this solution: having an environment where the test workload can be distributed effectively is ideal.

Summary

It should be evident by now that what I've described is a trivial solution to a common requirement and is not intended to be a commercial-grade substitute. This solution does, however, have benefits for those of us seeking a portable, economical, and effective means for generating HTTP traffic without having to employ and train chimpanzees – or leave the country in hurry. ☛

marc.connolly@oracle.com

AUTHOR BIO

Marc Connolly is by trade a programmer. Currently working for Oracle Corporation, he has worked for various companies, large and small, over the past 20 years. His focus has been primarily in product development for and with relational databases with occasional forays into stranger venues.

Next Month in *JDJ*...

The Java Message Service

What is JMS and where did it come from?

by Dave Chappell

Using the Java Platform Debugger Architecture

by Tony Loton

Power JMS

Applying the facade pattern to JMS using a custom protocol handler

by Tarak Modi

Product Review:

DeployDirector 1.3

by Sitraka Software (formerly KL Group)

by Joe Mitchko

How to Build a Telephone/Voice Portal with Java Phonelets

Part 1 of 2

by Kent V. Klinner III and Dale Walker

Integrating CORBA and J2EE

Integration servers help fill the gap



WRITTEN BY
PAUL MOXON

This article outlines how organizations can build and maintain an integrated system infrastructure to support their business needs using open industry standards, such as CORBA, Java 2 Enterprise Edition (J2EE), and XML. In doing this, they can maximize investments in existing technologies and integrate them in a powerful and flexible way into new e-business systems.

Today most organizations are trying to come to terms with the new age of the Internet and e-commerce. Large established companies see this as necessary for survival, while smaller companies view it as an opportunity to expand in markets that aren't confined by geographical location or budgetary constraints. The Internet and, increasingly, wireless communications (telephony or otherwise) are considered the future of business. Companies must be able to embrace this new economy to survive or prosper. Increasing reliance on the Internet is driving the uptake of technologies that support the development and deployment of Internet-enabled applications.

Predominant is Java. In all its forms, Java is clearly the preferred programming language of the Internet and, in the form of Enterprise JavaBeans, is now set to dominate server-side applications.

J2EE, which encompasses Enterprise JavaBeans (EJBs), JavaServer Pages (JSP), Java Mail, Java Servlets, and so on, is the governing standard for Java-based, Internet-enabled applications (although proponents of Microsoft's .NET architecture may disagree with this generality). The numerous J2EE Application Server products now available provide excellent tools and support for developing and deploying new Internet-enabled Java applications. These products are likely to become even more innovative and usable as vendors compete for mind and market share in what's currently a very crowded marketplace.

Although J2EE application servers are excellent products in their own right, they're not the silver bullet their vendors would like us to think they are. The J2EE view of the world assumes that everything is developed in Java – which is great if you're developing new systems from scratch. But who's really in this position? Who can simply ignore and throw away the many years of effort and money spent building existing applications?

As we all know from the problems encountered by "e-tailers," a fancy front-end Web site is no longer good enough. Web sites must be fully integrated with the back-end business systems such as order processing, warehousing, logistics, customer care, and so on in order to fulfill the customer's requirements quickly and effectively. In the vast majority of cases these systems are not written in Java (or as EJB applications) and therefore don't fit easily into the world of J2EE.

The question facing enterprise architects is: How do you integrate the new EJB applications with a legacy COBOL program running on an IBM/390? Before we try to answer this, we'll look at another integration technology.

CORBA

CORBA was designed for integrating applications in heterogeneous environments, that is, environments containing applications written in different programming languages running on different operating systems and different

machines. In other words CORBA was designed specifically to address the integration problems faced in a typical IT environment!

CORBA achieves this level of interoperability by specifying program interfaces in an implementation-neutral interface definition language (IDL). IDL can be mapped to almost any required programming language (C, C++, FORTRAN, COBOL, Ada, etc.), allowing you to develop your programs in the language of your choice. CORBA also specifies the Internet Inter-ORB Protocol (IIOP) as a common communications protocol for TCP/IP networks. Just as IDL hides programming language differences, IIOP enables different applications to communicate with each other through a standard protocol and hides platform and network differences.

Because CORBA is a much more mature standard than J2EE (CORBA has been around for more than 10 years), it has addressed many of the issues facing the integration of enterprise-scale systems. For example, allowing applications to interoperate using just the abstractions provided by IDL and IIOP isn't true integration. True integration occurs when applications can participate in the same transaction, share the same directory services, and use the same security model. For example, if CORBA-based applications are to participate in the same transaction, they need to know the transaction context. Fortunately, IIOP is designed to implicitly propagate transaction and security context information, allowing remote objects to share this information.

The OMG has specified a number of additional services to support the core CORBA specification, including an Object Transaction Service (OTS), a Security Service, a Naming Service, and an Event Service (now superseded by the CORBA Notification Service). These form a framework of supporting services that are essential for large-scale systems and are designed to promote true integration between applications rather than just interoperability.

and simpler environment for developers. CORBA is difficult for many developers, one of the main hindrances to its mainstream adoption. However, CORBA's weaknesses – such as the lack of a true framework to support the rapid development of applications – are the strengths of J2EE. Together they make a formidable package, as can be seen in Table 1.

Now return to the question facing enterprise architects: How do you integrate your new EJB application with a

performance when the system works at the speed of the slowest component, or even deadlocks – applications end up in a gridlock of requests waiting to complete before subsequent operations can be processed.

An alternative architecture that avoids the pitfalls of closely coupled systems is an asynchronous – or loosely coupled – system architecture. This architecture decouples the applications from directly interacting and uses a messaging system as the “buffer.” The messaging system supports guaranteed delivery semantics (also called “store and forward”) to ensure that messages are received by the target applications. This means that an application can send a message, then continue to perform other work, confident that the messaging system will ensure delivery to the intended recipient.

An application sends data to the messaging system and then gets on with its own internal functions – it doesn't need to know what other applications receive that data, what methods are in their APIs, or even if they're currently online. Similarly, changes can be made to an application and its interface to the infrastructure, or new applications can be added, without affecting others.

For example, consider the situation in which one application needs to send data to 10 other applications. Using traditional EAI or just an ORB, the sending application's API needs to understand the API “data write” methods of all 10 receiving applications to write the data. If a receiving application's API changes, the method calls in all sending applications may also need to change. Using messaging, the sending application needs to know only one method: how to send data to the messaging bus. If other applications fail, change, or are added, nothing changes for the sending application.

This makes the overall system much more flexible and resilient. To add another application the system is required to be integrated only into the messaging architecture rather than directly to each application that it interacts with, reducing the number of integration points dramatically. The decoupling also reduces the effect on the system when an application fails. As the messaging system acts as a buffer between applications, the remaining applications can continue to operate and any messages destined for the failed application will be queued until it's restored. Finally, with the messaging system acting as a buffer between the applications, they can all work at their optimum speed. A slow application

	CORBA	J2EE
Distribution	Strong - traditional CORBA area	Weak - RMI not scalable - now use RMI-IIOP
Server Framework	Weak	Strong - but Java only
Infrastructure Support	Weak - but comprehensive set of services	Strong - leveraging CORBA Services

TABLE 1 J2EE and CORBA

The Best of Both Worlds

Fortunately Sun recognized the strengths and advantages offered by CORBA and incorporated them into J2EE. The J2EE specification requires compliant application server products to support RMI over IIOP as a transport protocol, in addition to the Java-centric Java Remote Method Protocol (JRMP). With the implementation of RMI-IIOP, Java RMI objects can be accessed by CORBA objects written in another language. Moreover, Java RMI objects can access CORBA-based applications regardless of what programming language they were written in. This brings the strengths of CORBA – cross-language integration – to J2EE. J2EE Application Servers also benefit from the industrial strength of IIOP when used in large-scale systems. It's now widely accepted that IIOP scales much better than JRMP, and therefore provides a better protocol on which to base enterprise developments.

J2EE support for transactions is provided by the Java Transaction Service (JTS). The J2EE specification currently recommends (in version 2.0, it requires) the support of JTS layered on top of the CORBA Object Transaction Service. The use of OTS, and hence IIOP as a communication protocol, supports the interoperability of transactions between application servers and legacy applications.

J2EE leverages the strengths of CORBA to make J2EE application servers more robust and scalable and, at the same time, provides a more productive

COBOL program running on an IBM/390? With the implementation of RMI-IIOP, the EJBs can access CORBA-based applications regardless of what programming language they were written in. So we can use an ORB that supports COBOL (e.g., IONA Technologies' iPortal for OS/390) to provide the interface to the COBOL application, and the EJB can use RMI-IIOP to interoperate with the application's CORBA interface and, via this interface, the COBOL application itself.

Therefore, the combination of J2EE to develop new Internet-enabled applications and CORBA to integrate these applications with existing systems, provides a complete solution for integrating enterprise systems...or does it?

Synchronous vs Asynchronous Architectures

The type of integration we've discussed so far has been based on synchronous interactions. That is, a request is made to the target application (in the form of a remote method invocation) that processes the request and returns any results. This request-reply mechanism is simply the distributed form of a method call to a local Java object. It works well within a Java program contained in the confines of a single JVM, but doesn't scale well when it spans multiple JVMs or even non-Java applications. A synchronous – or closely coupled – architecture can become fragile when it's made to grow this way. Applications become prone to slow perfor-

won't affect the performance of the others, because the messaging system will store any messages until the slow application can process them.

Despite these benefits of greater flexibility, scalability, and robustness, asynchronous architectures aren't the solution to all problems. There are circumstances in which a synchronous request-reply mechanism isn't just the

This scenario requires a synchronous request-reply interaction; an asynchronous interaction wouldn't be acceptable to the customer, who needs to receive transaction reference information (order number, transaction ID, etc.) straightaway. Subsequent fulfillment of the order may, however, be processed in an asynchronous manner; this doesn't concern the customer, who has now

architecture, which has led to a new generation of products called *Integration Servers* or *Brokers* that complement J2EE Application Servers.

Integration Servers

Both application servers and integration servers are frequently used for integration. Application servers tend to be used by developers building some new part of an application's functionality. They support a closely coupled architecture using a request-reply interaction and tend to support fine-grained components. Integration servers, on the other hand, are focused toward integrating existing functionality rather than developing new applications. They're based on a loosely coupled architecture using asynchronous messaging and support coarse-grained integration. In this sense, application servers and integration servers can be seen as complementary (see Figure 1). The application server provides a core deployment platform and tools for application development, while the integration server augments this with tools for application and data integration, architecture abstraction, and technology and vendor "hiding" layers. In most nontrivial projects, both types of product will be required.

Figure 1 also illustrates the main functional components of an integration server, namely:

- **Message broker:** Provides powerful interapplication messaging functionality across multiple messaging models (e.g., CORBA, EJB, SOAP) and vendor (e.g., IBM, TIBCO) solutions.
- **Data transformation:** Makes the complex task of reconciling and using disparate data formats much easier.
- **Process modeling and automation tool:** The essential link between an end user's business processes and OEM's software applications and IT infrastructure.

Let's look at these main components in more detail.

Message Broker

This is the "hub" of the integration server. The message broker provides the standards-based asynchronous messaging capabilities that are essential in a loosely coupled architecture. The key standards for the message broker are CORBA, EJB (JMS), and SOAP. The message broker must also integrate to other messaging systems, such as IBM's MQSeries and TIBCO/Rendezvous, so that legacy applications using these products can be integrated into the enterprise system.

The support for standards-based messaging APIs allows developers to use

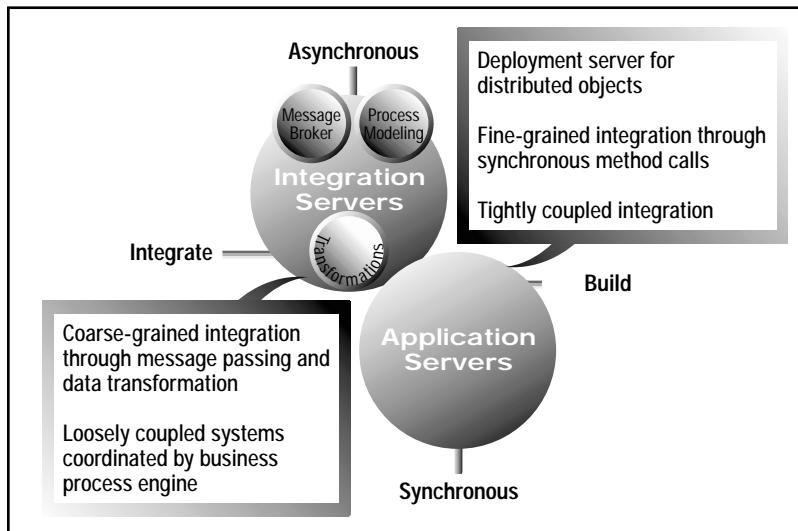


FIGURE 1 Integration servers and application servers

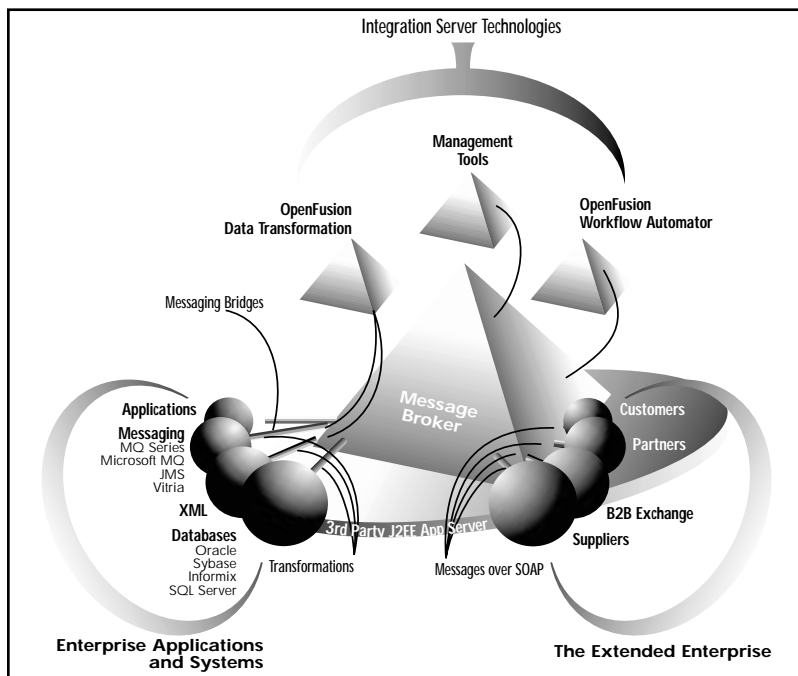


FIGURE 2 Integration server

best solution, it's the only acceptable one. Consider the case of a customer making a purchase over the Internet using a credit card. The resultant transaction must be processed there and then, so the customer knows that the purchase has been successfully completed and the vendor has the security of a completed transaction.

received the required reference information in case of a problem later.

Therefore, a typical enterprise system doesn't consist of only closely coupled synchronous integration, nor will asynchronous messaging-based architecture meet all the requirements of such a system. Enterprise-scale systems require a mixture of the two types of

their messaging API of choice. Therefore it should be possible for an EJB to send a JMS message via JMS, and for this message to be received as an MQSeries message by an MQSeries application. The sender should be unaware of the message format required by the recipients, and the recipients should be equally unaware of the format of the message when it was sent. The message broker is responsible for ensuring that the message is transformed into the correct format before it reaches its final destination.

The message broker must also support the usual guaranteed delivery ("store-and-forward") semantics and provide content-based routing according to business rule.

Data Transformation

The role of data transformation is to ensure that when data reaches its destination (usually via the message broker) the format of the data is what's expected by the receiving application. The rise in popularity of XML and the "opening up" of ERP and CRM applications to support integration via XML will make data transformation capabilities even more critical to enterprise systems. It's quite possible that two-partner organizations will support different data models (whether traditional data models or XML schema) within their systems and that data transformation will need to be performed if the organizations are to integrate their systems.

Process Modeling and Automation

The final major component of the integration server is process modeling and automation. The process modeling allows business analysts to model the business processes and define the various business rules and activities that drive these processes. The process automation allows the analyst to map these business process models onto the underlying applications. The integration server then coordinates the operations of these applications according to the defined business rules.

The process automation tools must also be standards-based and support interaction between applications that are based on different architectures (e.g., CORBA, EJB, DCOM).

The architecture of a typical integration server is illustrated in Figure 2.

The use of messaging and data transformation also lends itself to supporting XML as a data format. XML has rapidly become the de facto data format of e-commerce, but has just as rapidly become fragmented by the various organizations that are defining business schema for XML data integration (e.g., BizTalk, RosettaNet, OASIS). This has resulted in a number of different forms

for representing common business entities, for example, purchase orders. This disparity of forms requires transformation technology to change the data into the expected format, and the combination of asynchronous messaging and dynamic data transformation of XML make this a relatively simple but effective process. This results in the ability to integrate with applications, not just within the enterprise boundaries, but also with applications that are external to the corporate firewall (i.e., applications that exist in the "extended enterprise" – suppliers, partners, and customers that form part of the corporate supply chain). Integrating these applications using J2EE and CORBA alone would be an extremely difficult, if not impossible, task. But integration servers used with the J2EE application servers now make this type of integration more manageable.

The Future – Java Connector Architecture

Sun recently released the proposed final draft of the J2EE Connector Architecture (JCA) 1.0, aimed at providing a standard way for back-end applications to plug into the J2EE Application Server platform. JCA, which was developed under the Java Community Process (JCP), will be part of the next version of J2EE, version 1.3, expected in early 2001. Although JCA is missing some key functionality required in more complex integration situations, it marks an important step toward reducing the costs and burden of back-end integration.

The J2EE Connector Architecture defines a set of functionality that application server vendors must provide and that back-end system vendors (e.g., SAP, PeopleSoft, Siebel, Clarify, or third-party connector developers) can use to plug into the application server. The architecture doesn't specify how this capability is implemented; that's up to the platform provider or the connector developer.

The JCA has two basic components: the Common Client Interface (CCI) and a set of system-specific services. An adapter developer provides an interface to CCI along with its side of the system contracts specified as part of the connector architecture. The application server vendor implements its side of the system contracts as part of its base J2EE platform.

The JCA provides the following capabilities to the application servers:

- **Transaction management:** Enables the transaction manager provided within the EJB application server to manage transactions across multiple back-end systems.
- **Connection management:** Enables

the application server to create and manage connections to back-end systems. One important capability provided is support for connection pooling, since connections to back-end systems are expensive.

- **Security:** Enables the developer to define security between the EJB server and the back-end system. The specific security mechanism used is dependent on the security mechanism provided by the back-end system.

Despite these features, JCA 1.0 is still a point-to-point solution for integrating applications. It doesn't support asynchronous communications and it only supports the synchronous request/reply model. This means that a JCA resource adapter can call a remote system and wait for a response, but the remote system can't initiate a call back to an adapter at a later point. Although this is common in the application server world, it's not well suited for more complex integration scenarios.

It's unlikely that JCA adapters will be available until the latter half of 2001 at the earliest, and even then will be based on the version 1.0 specification. Thus they won't support asynchronous communications between the EJBs and legacy applications. It'll probably be well into 2002 (or later) before asynchronous adapters are available. Until then, integration servers look to be the best way of providing this loosely coupled integration between the Internet-enabled applications and the existing back-end systems.

Conclusion

The rapid acceptance of J2EE has done much to promote the use of the Internet as a viable e-commerce medium. However, J2EE has weaknesses when it comes to building scalable and resilient enterprise-wide architectures. By leveraging some of the features of CORBA, J2EE does overcome some of its limitations. However, it's still not enough to satisfy the real-world requirements facing most organizations. The recent emergence of a new generation of standards-based products – integration servers – will help fill the gap in the J2EE-CORBA combination. The integration server products introduce a more flexible and robust architecture based on asynchronous messaging to J2EE. This helps it extend beyond the boundaries of the corporate firewall and provide integration across the Internet using technologies such as XML and SOAP. ☉

paul.moxon@prismtechnologies.com

AUTHOR BIO

Paul Moxon is a product manager at PrismTech Ltd., a software company specializing in systems integration – CORBA and J2EE integration in particular. Paul has more than 10 years of experience in various distributed systems, including DCE, CORBA, DCOM, and J2EE. He graduated from the University of Northumbria in the United Kingdom with a degree in business administration.

Representing
data in
automated
systems

Business Rule

Java Objects

WRITTEN BY **KEN MOLAY**

In May and July of 2000, **Java Developer's Journal** (Vol. 5, issues 5 and 7) ran a two-part article on how business rules can be implemented in Java. To recap, business rules are a formalized representation of the policies, practices, and procedures of an organization, describing how business should be conducted under any particular set of conditions. Business rules aren't a programming concept but rather a business concept. The business rules of an organization may be contained in policy manuals, memos to employees, unwritten "tips and tricks" passed from employee to employee, or lines of program code spread among various applications serving different business needs.

The use of specialized business rule authoring/execution environments with independent rule repositories was introduced as a way to gain more control, consistency, and reuse of business rules throughout the automated systems of an enterprise. Business rules can be applied to Web-based interactive systems, and this is often the easiest way to grasp representative application examples. For instance, Web-based merchandising rules may decide the best products to display and pro-

mote to an individual shopper and the proper pricing and discounts to offer on an order.

However, many other uses of business rules affect back-office processing or interactive system behavior via communications channels such as automated phone systems, electronic kiosks, wireless devices, and live customer agent support. (These channels are often referred to as "touchpoints" since they describe various means by which an end user can "touch" the enterprise.) It's also valuable to remember that rules may affect interactions not only with customers of an organization, but with its employees, suppliers, business partners, and the general public.

The previous **JDJ** articles mentioned products such as Blaze Advisor from Blaze Software (now a part of Brokat Technologies), JRules from ILOG, and Jess from Sandia National Laboratories as tools companies can use to implement business rules in Java applications. This article explores the representation of business rules and data objects, and focuses on how rule objects interface with external systems. We also examine issues in usability for nontechnical employees charged with updating business policies and operational logic in their organizations.

Using Objects in Rules

As shown in previous articles, business rules can easily be thought of in terms of IF-THEN declarative statements, pairing a specific condition set with a desired action:

```
IF Amount of Order > $100
THEN Discount = 5%.
```

Although this is an arbitrary representation, many programming languages have made the analogy and it seems to be one that maps well to

human understanding of situational responses. Both conditions and actions must be specified in terms of objects, a concept familiar to Java programmers, but one that's often confusing to business people attempting to define or maintain business-rule logic for an application. It's easy for Java programmers to forget that the term "object" is used in a conversational English sense to specify a tangible item that can be seen, touched, and manipulated. Properties of objects are hardly ever thought of by nonprogrammers as creating different instances of a single object type.

For instance, to the layman, a yellow tennis ball and a white tennis ball are two distinct objects, each having a physical manifestation independent of the other. Making the conceptual leap to each being an instance of a single object type with varying property values is a training exercise. It gets even worse when dealing with intangible process concepts, such as "the ExecutionPriority of a CollectionEvent."

Thus dealing with object and property definitions in business rule systems is a two-sided problem. Any company using formalized systems will already have created object definitions to represent the data being manipulated. Installing a new piece of software, such as a business rule engine, requires the programming staff to define the company's existing object representations in the proprietary framework of the new software. And because business rules are intended to be exposed to the business users, those users must have a familiar way to understand, use, and manipulate these objects.

Most corporate IT departments are responsible for a variety of systems written in different languages, incorporating various third-party software packages produced at different times, and making use of object definitions originating in Java, databases, CORBA, COM, XML, and other formats. An attempt to rely exclusively on any one of these object representations by a new vendor is likely to cause implementation difficulties and delays. The approach favored by many successful business rule software vendors is to create a proprietary internal object representation and use utilities to map to external business object models.

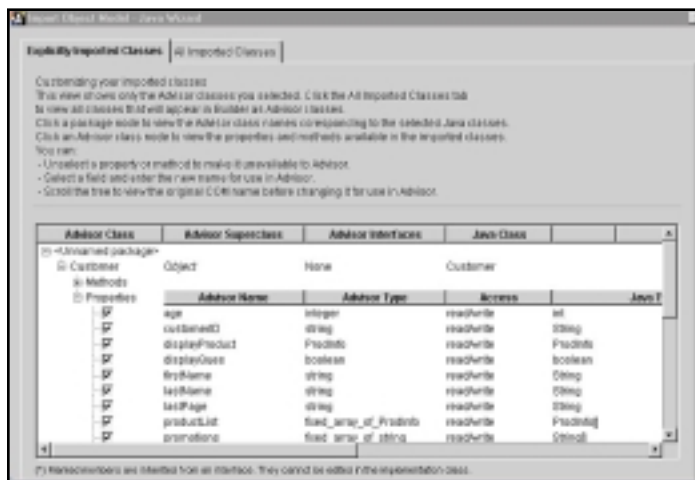


FIGURE 1 Blaze Advisor object model import wizard

As a demonstration of this approach, Figure 1 shows an object model import wizard from Brokat Technologies' Blaze Advisor.

The utility has been given the name of an external Java class and has generated a table showing all the object and property definitions the class contains. The implementation specialist may include or omit individual properties (and Java methods, if desired) to be made referable from within the business rules. Individual properties may be renamed for use within business rules in order to make them more convenient to write or more understandable for business use.

Importing external object model definitions and mapping to internal representations enhances efficiency and speed in implementing the new business rule system. Object models need not be manually printed out, checked for type definitions, and recoded by hand in a proprietary language. Organizations looking to integrate business rules into their operations should look for smooth mapping and integration facilities for whatever object model representations they use.



FIGURE 2 Blaze Advisor database import wizard

A similar object model mapping can be used for external database definitions. In the Java world, JDBC connections interface with database tables (see Figure 2).

Columns may be identified from a single table or from multiple tables, and SQL-formatted WHERE statements can be used to identify specific instance inclusions, joins between tables, and other data restrictions (see Figure 3).



FIGURE 3 Using SQL to restrict data instances

Once object/property definitions have been imported, business rule authors may wish to create subclassed objects for use within their busi-

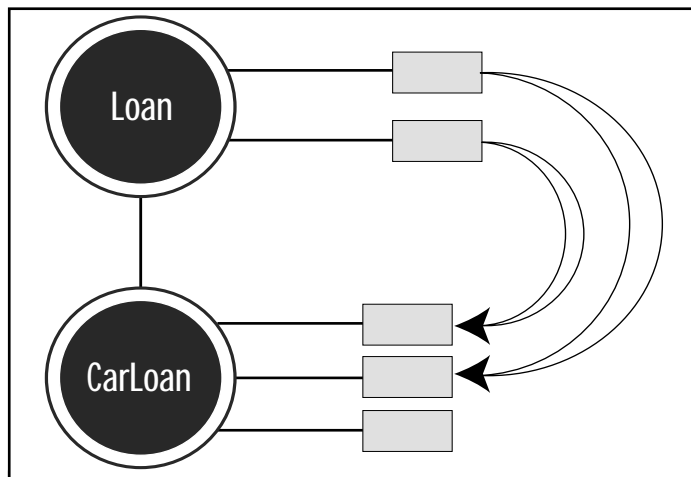


FIGURE 4 Subclassing objects

ness rules. For instance, a finance company may have a Loan object defined in their general business systems. But for writing business rules dealing with automobile purchases, they want to define a special CarLoan object, inheriting the properties of Loan (see Figure 4).

This may be accomplished through graphical editors, filling in new property names and types in a form template, or through simplified code constructs such as:

```
A CarLoan is a Loan with
{ an odometerReading : an integer,
  a dealerName : a string }
initially { odometerReading = 0 }
```

Linking External Applications to Business Rules

Once objects have been defined for use in the application's business rules and mapping has been made to external business objects, a runtime link must be established to keep internal rule assertions and external data synchronized. This takes two forms: data being passed to the rule engine for use in reaching rule-based conclusions, and data being updated in external systems as the result of rule execution.

Remember that a business rule service is one component of an overall application. Depending on the application, the rule service may be explicitly called in order to make a logic-based decision, monitor events, and wait for a trigger to stimulate rule firing, or to create an external event to obtain additional information needed for processing. With mapped data objects, the business rule engine can interact in any of these sample use case scenarios:

1. **Application:** Instantiates data in a *customer* object and a *loan application* object. Should I approve, decline, or refer the application?
2. **Rule Engine:** Return an answer of approve/decline/refer.

1. **Application:** Starts the rule service to monitor external *competitor's price* object.
2. **Rule Engine:** Whenever *competitor's price* changes, fire rules to potentially update our *price* and *discount* business objects for use by all systems.

1. **Application:** Updates *customer age* object value and continues rule processing.
2. **Rule Engine:** To continue processing, a value for *customer age* is required. Retrieve the value from an external database table or have the application prompt the user.

Business Person Interaction with Rule Objects

The preceding sections dealt with the mechanical linkage of objects in a business rule package with external data systems. These are concerns for the technical implementation personnel at a company. Once the IT department has created definitions and linkages for their business objects, the question becomes: How can the objects be manipulated to drive business processes? It's then an issue for the business-level personnel at the company.

System development and maintenance has traditionally taken the form of IT personnel creating system requirements definitions through a process of interviews and cooperative work sessions with the business users of the proposed system. As development progresses and the system becomes more tangible and well understood, the business people often think of new exception cases, additional uses in new scenarios, or refinements to their original logic. Integrating logic changes into system programming can be laborious and time-consuming. Once the system is completed and in

“
The differentiator
becomes the
ease, speed, and
security with
which rules can
be changed by
nontechnical
professionals
”

place, external and internal business factors may require periodic updates to the business logic in the form of new decision conditions and results.

Examples are endless. Merchandisers may vary product discounts based on supply and inventory considerations. Financial institutions may change interest rates or information requirements based on new regulatory mandates. Human Resources policies may affect employee benefits based on organizational changes or management discretion.

In all such cases a delay between making the business decision and having it reflected consistently throughout all corporate information systems is costly, frustrating, and sometimes even legally actionable. IT departments may be called upon repeatedly to make small changes to systems, taking their concentration and resources away from other system development tasks. Conflicts arise with IT managers trying to create structured work processes and release schedules while business management attempts to react swiftly to new business conditions.

Business rule systems were created to centralize the business logic aspects of automated systems and

separate them from the underlying infrastructure of the systems' operation. In this way, rules can be changed to control conditional actions without affecting the overall programming code. If the rules are then exposed to the business decision makers themselves for maintenance and management within a constrained range of control, the IT department can be freed for major structural changes while the policy makers manage day-to-day business logic.

The structure for facilitating such logic separation has been implemented by most business rule software vendors. A centralized repository of rules, visible and alterable by the business owners, is common in such systems. The differentiator becomes the ease, speed, and security with which rules can be changed by nontechnical professionals.

Business rules are usually expressed in some structured language unique to the software vendor. As an example, in Jess from Sandia National Laboratories, a rule might be expressed in the following form:

```
Jess> (defrule CanPersonVote
      (person (age ?x))
      (test (< ?x 18))
      =>
      (printout t ?x " is a minor and may not vote"))
```

The problem with such formalized languages is that the personnel require programming training in order to make changes, and there's not an intuitive conceptual link between the business concept and the objects, values, conditions, and actions that will be performed.

A business rule language should offer flexibility in authoring to suit various users' skills and comfort levels and should provide a natural link between a business concept and its formalization in the system. As an alternative example, here's the same rule code expressed in Blaze Advisor's structured rule language:

```
Rule CanPersonVote is
If person's age is less than 18
then print(the name of the person " is a minor and may not vote").
```

Note the natural-looking sentence structure of the rule and the two different ways of referring to the object – *person* – and its properties – *age* and *name*. In classic Java dot notation, the object-property combinations would have been written as *person.age* and *person.name*, and indeed, the product allows for this usage in the syntax. Mathematical comparison operators can also be expressed either in symbolic form familiar to programmers or in English language formats. To illustrate the flexibility of the language, the following statements are all well-formed constructs and are functionally equivalent:

```

If customer.age > 16 then...
If the age of the customer is more than 16 then...
If customer's age exceeds 16 then...

```

There's no performance penalty for using one version versus another, as they're all compiled into the same internal runtime representation for use by the rule engine.

Rule Maintenance Without Code

Even with more natural representations of business concepts in a business rule language, maintenance of the rules requires training in the specifics of that language, the use of some form of text editing environment, and exposure of the full rule syntax to the business policy maker. Maintenance personnel must also know the correct names of any objects and properties they wish to use in a rule. No matter how natural the names are, they can be overwhelming when dozens or even hundreds of names are sprinkled throughout a complex system. Business maintenance personnel have access to the entire rule, which could lead to undesired changes in the structure of the rule code, unauthorized changes to values they should not be altering, or simple mistakes due to unfamiliarity with the specific object and property names, or even from typographical errors.

The challenge for systems personnel is to build a stable system and then allow a specific subset of ongoing maintenance changes by the business units. Change should be controlled with authorizations, and the change process should insulate the maintenance personnel from process complexity and memorization.

One approach is to allow rule developers to explicitly identify pieces of rules that should be exposed to business users for maintenance. These objects, conditions, or values may be constrained to a predefined set of allowed values, or may be left open for unrestricted input. Web pages are constructed using terminology and graphics that are familiar to the business policy makers, with replacement values presented as input fields. These may use fill-in-the-blank boxes or HTML features such as pull-down lists or radio buttons for entry. The use of password-enabled, Web page access security determines maintenance access, and rule changes can be tracked and authorized against an LDAP rule repository for check-in and check-out of rules.

As an example, let's use a generic merchandising rule:

```

Rule ProductDiscount is
If customer's age is more than 55
then discount = 0.05.

```

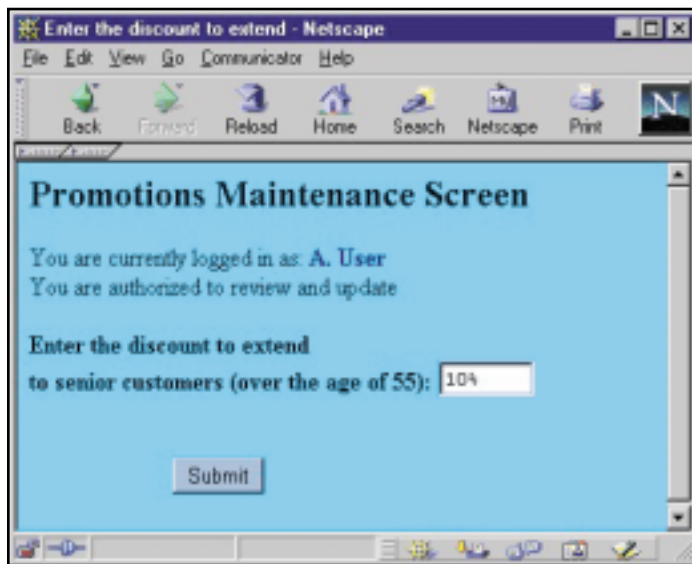


FIGURE 5 A simple rule maintenance screen

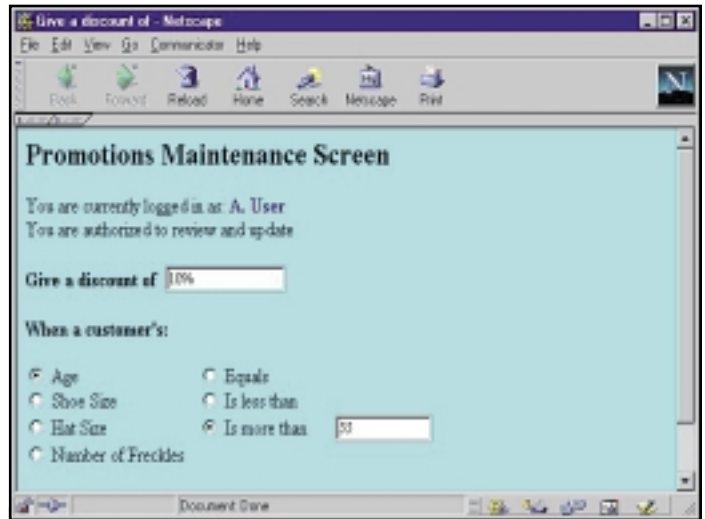


FIGURE 6 A multichoice rule-maintenance screen

In the simplest form of maintenance, we might want to let the business unit change the amount of the discount given to our senior customers. We highlight the discount value and place it on a Web page, along with text that describes the use of the value. Note that we don't have to use any of the syntax of the rule itself. A JSP tag links the value entered on the Web page to the replacement point in the rule code (see Figure 5).

In a more complex example, we might make a generic template for defining discounts based on any one of a number of conditions:

```

Rule ProductDiscount is
If customer's age is more than 55
then discount = 0.05.

```

We can let the policy maker base the offered discount on one or more predefined properties for the customer object. (Let's hope no merchandiser would actually base discounts on the criteria shown here!) He or she can also define the type of comparison operator, the comparison value and, of course, the discount amount. The rule-maintenance Web page might look like the one in Figure 6.

With a system such as this in place, maintenance personnel no longer have to remember proprietary syntax or object/property names. They can think purely in terms of the business decisions they're making. System developers can rest assured that the maintenance person can't make unauthorized changes, such as basing discounts on the customer's height or weight, or setting the action to apply a surcharge instead of a discount.

Summary

Business objects are a standard means for representing data in automated systems. Any company installing a business rule system should make sure that importation of existing object models is simplified and accelerated so they can quickly move to the core business task of writing the operational business rules. For business people to take advantage of the centralization and user maintenance of rules, they must be able to understand the objects and properties used in the business rules. Representation of rules should allow for flexibility in expression of these objects and companies should make sure that rule maintenance is practical on an ad hoc basis. This requires protecting the integrity of the rule structure while allowing business policy makers to make authorized changes in a comfortable and familiar manner without programming or training in specialized tools. ●

AUTHOR BIO

Ken Molay, director of product evangelism for Brokat Technologies, a Brokat company, has more than eight years of experience in development, management, and customer application of expert systems and business rules.

ken.molay@brokat.com

Things Can Only Get Better



WRITTEN BY
ALAN WILLIAMSON

As I write, it's the day after Boxing Day. I was supposed to have this written yesterday, but to tell the truth I just couldn't find my muse...so today will have to do. As I wake up this morning I see in horror the latest on a madman who let loose with a firearm on his work colleagues. Sadly, nothing terribly new in that fact for a nation that's so proud of its right to bear arms. However, this particular instance had it occurring within an Internet consultancy company. This industry is no longer as safe as the career officer once promised.

Over this festive period we've been fed the usual nonsense on television. This year, however, I've been avidly glued to a documentary series chronicling the American West and how it was "won," and I use the word *won* in the loosest sense. In our schools we aren't taught that much American history; we stick to castles with kings and queens lopping each other's heads off. This series has whet my appetite enough to dig for more information. I have a lot of unanswered questions. So if anyone knows any URLs of reliable historical reference, please e-mail them to me.

Enough talking about the past and more about the future. This column, after all, is Industry Watch, as opposed to Industry Watch of Yore!

TagFusion

I've never been a great fan of JSP (JavaServer Pages) and what have you. But everywhere I look it seems that all the major application server vendors are throwing a serious amount of development and marketing behind this (and I will probably upset some JSP purists here) god-awful technology. I just can't see the allure of a technology that gives so much power to those that should never be entrusted with a calculator, let alone a "Webplication." Sure, the concept seems interesting enough, providing the ability to embed Java code within an HTML page. Sounds great on paper. However, ask anyone who's had to support someone else's JSP abortion, and you get a different perspective on the whole JSP debate. It's a support nightmare.

Now it would be wrong of me to dismiss JSP completely without giving it a proper hearing. The custom tag API in which it was introduced is indeed a great piece of technology and one that should be exploited a lot more. While I'm interested in seeing where this one is heading,

I'm not at all convinced it should be anywhere near JSP. Doesn't fit as far as I'm concerned. A custom tag library is probably closer to that of the servlet API as opposed to the JSP interface. The purists will argue that a JSP page is just a servlet. Technically, I can't argue with them on that score. However, JSP is yet another layer slowing down the page-rendering process, where a layer need not exist.

The reason I say custom tags are closer to the servlet API is simple: the servlet API has had this facility from day one, with the Server-Side-Includes (SSI) facility. So why reinvent the wheel?

With SSI we had the ability to completely hide our inner workings from those who might meddle. We simply presented them with a simple interface. An interface they were used to using. An interface that wasn't foreign to them. A - dare I say it? - tag!

JSP, I believe, blurs the border between presentation and business logic a little too much. It allows those who know nothing about the business logic to fiddle, and those who haven't an artistic bone in their bodies to meddle. A very ugly scene can ensue.

I find it wonderful that Java is so accepted now. It's refreshing to arrive on a client's site and find their server already Java enabled. Sadly, this wasn't the case four years ago. Not only did you have to install your own software when you arrived, but also that of the JVM, the servlet engine, and sometimes even the Web server.

Early on at n-ary we set about developing our own HTML tag templating system. We quickly discovered how useless it was to write Java servlets for each client project. The Java servlet is indeed a clumsy technology when all is said and done, and in the majority of cases developers are creating more work for themselves than they're supposedly solving.

To this end we designed a complete templating system that runs on top of the

Java servlet API. Our tags were extensive and very versatile and, thanks to Java, ran on all servlet-enabled platforms. Over the next three years our system, which we called *tagservlet*, was under constant development with testing in the field.

This time last year we were about to release it free to the world. However, as we looked around in this very busy marketplace, we pulled back and thought, The world doesn't need yet another HTML template system thrust upon it.

What a difference a year can make.

We studied the alternatives closely and decided we would adapt our tag system to fit alongside that of another. The one that resembled our tag structure and that our clients would moan about was CFML from Allaire. ColdFusion, they would say, "Great tag system, but slow as hell." We spent a month or so looking in depth at this system and decided to change our *tagservlet* to what is now known as *tagFusion*.

The tag switchover didn't take as long as we first envisioned, with the majority of time devoted to developing the expression library that CFML boasted. We launched our new and improved *tagFusion* to our clients in beta form in the summer of 2000. It was received very well, outperforming the ColdFusion server in the area of six to 10 times for some pages. We went through a series of rigorous tests and ironed out the majority of the wrinkles.

We started shipping CFML solutions to our clients, creating the necessary custom tags that their particular requirement warranted. We were no longer a Java software company, but a solutions company. It was a spooky changeover for us.

Would you believe we had to downsize our Java team? We had to make some of our Java team redundant simply because we didn't have the work to support them. We had created a tool that was so successful for our consultancy firm that it changed the focus of our entire business.

We then began hiring other skill sets to support the emerging market that



tagFusion opened up for us. Not only are we now able to take on existing Java projects, but we can deliver them faster with a standard that our clients feel safe with without compromising the openness and deployment features of running under a J2EE environment.

One of the first pieces of advice given to me when I set out to build a company was not to be scared to change direction. I've seen many large corporations changing direction and moving into areas they're not customarily known for. Take HP, IBM, Sun, and even Microsoft as a few examples. But I had just never imagined n-ary being anything other than a pure Java company. But here we are, starting 2001 as a new company that doesn't focus its energy entirely on Java.

We effectively coded ourselves out of a job. But I'm not sad. I'm excited at the opportunities our vision has brought and will continue to bring. We haven't completely left Java behind, though. We have a supporting team for tagFusion, and I can still be found cursing and swearing at CodeWarrior!

Digital World

Technology is quite wild in the parts of our lives it manages to reach. My son was born recently, and within four hours of that event, thanks to a digital

camera and a Web site, I had all of our family and friends witnessing the new addition to the clan, irrespective of their distance. It's now nearly a month since Cormac introduced himself to the world, and already I have more photographs of him than my parents have of me in my entire child life. Add to that the number of RealVideo clips I've made of him. I have to sit back and marvel at the ability to do all this. What really impresses me is how little it cost to produce this technical diary.

With WAP just beginning to make an appearance, I have to wonder what tools will be at Cormac's disposal when he goes to document the new life of *his* first child. Once we have that figured out, we have to start coding for those tools. What ingenious ways can we find to pull as much functionality out of this hardware with just software? This is what excites me about this time we live in. We're young enough to remember what it was like in predigital days to appreciate the move to 1's and 0's.

To us the change is far more dramatic. My son will never know the wonders of a vinyl record or understand the complexity of the classic Space Invaders. I can't even imagine what the game of his day will be. I just hope I'm about to witness some of it and, more important, together with him, make some of it happen.

Original Drive!

I do a lot of reading – mainly current affairs-type stuff. One of the chaps I follow is one Jesse Berst of ZDNet fame. Jesse, a bit like myself, isn't scared to court a little controversy once in a while, and I admire anyone who does that sort of thing. Takes a lot of guts to throw one's body over the barbed wire. Jesse produces a daily e-mail editorial that's highly informative and sometimes quite entertaining. However, on one such occasion Jesse was twittering about wireless and what have you, and wrote a sentence I thought I had heard somewhere before. On my weekly back review of *Wall Street Journals* I found the phrase again. Printed two days before Jesse's article. Coincidence? I don't know. You decide.

But rest assured about reading this column: it may on occasion be drivel, but at least it's original drivel!

And on that note, see you in the March issue of *JDJ* ☪

AUTHOR BIO

Alan Williamson is CEO of the first pure Java company in the UK, n-ary (consultancy) Ltd, a Java solutions company specializing in delivering real-world applications with real-world Java. Alan has authored two Java servlet books and contributed to the servlet API.

alan@n-ary.com

WRITTEN BY MANI MALARIVANNAN

Jlink: Cybelink's Framework for Creating Reusable Enterprise Components Using J2EE

When Sun released J2EE to capture the growing e-business market, it changed Java from a language to an enterprise platform.

Several key players such as BEA and Oracle have pledged their support and endorse J2EE standards in their application server products. Several other companies are either already using the Java application server or are thinking of using it in the near future. These companies are scrambling to come up with a scalable enterprise architecture that works with existing technology and also grows with future changes. However, developing scalable and adaptable enterprise architecture is a difficult and time-consuming task. Several software-engineering principles such as OO methodology and software patterns need to be used when constructing the enterprise architecture so it will last.

To solve this Cybelink created Jlink, a vendor-neutral framework based on Sun's J2EE Blueprint recommendations and guidelines. Architected at a high-level, Jlink can be easily adapted to work with any vendor-specific, Java-based application server such as WebLogic and WebSphere. In this three-part series we'll discuss the Jlink framework; in Part 1 we'll discuss the Servlet/JSP portion of Jlink and describe the problems associated with the Servlet/JSP technology. In Part 2 we'll describe the Jlink architecture and how it solves the Servlet/JSP problems. In Part 3 we'll describe the workings of Jlink and provide an example.

Web Programming Model and Servlet/JSP Container

Before going into the details of Jlink, we'll discuss the Web programming model and explore how the Servlet/JSP container supports it. Any Servlet/JSP-based Web application can be modeled using the following simple actions: a user submits a request using a Web browser and clicking a button or a URL link. The HTTP request first goes to the Web server and if it's plain HTML, it sends the HTML page to the browser. If the request needs to execute a Servlet/JSP, the Web server passes the request to the Servlet/JSP container, which processes the request by invoking the appropriate Servlet/JSP and sending a response back to the browser. The Servlet/JSP might process the request itself or send it to a JavaBean or EJB, extract the result, and transfer it to the browser via the Web server.

J2EE defines a Web container as a place where servlets and JSPs live, and it acts as a bridge between the client and EJB containers. The communication between the Web and EJB

Part 1 of 3

Exploring
Servlet/JSP
technology

containers is carried out by JavaBean components. A crucial aspect of Web development is to architect the middle tier, which processes the HTTP request and sends the results back to the browser. Jlink provides an easy mechanism to handle HTTP requests from the browser and send them to the appropriate JavaBean component, which processes the results and sends them back to the browser. An HTTP request can be processed in one of the following ways:

1. Within Servlet/JSP components
2. Within JavaBean components
3. Accessing the EJB business components using local calls or through RMI/IIOP
4. Accessing the Enterprise Information System tier (EIS) using a connector
5. Accessing databases using JDBC

This Jlink currently supports Options 1 and 2 and has been architected in such a way that future supports for Options 3, 4, and 5 can be integrated into it.

Servlet/JSP Container Runtime Environment

Before going into the details of our Jlink architecture, we'll discuss the Servlet and JSP working model in detail. Once we understand the inner workings of that, we can discuss the Jlink architecture.

When an HTTP request with a Servlet/JSP comes from a browser, the Web server routes the request to the Servlet/JSP container (see Figure 1). If the HTTP request comes for the first time, the Servlet/JSP container compiles the Servlet (in the case of JSP, it's converted into a Servlet) and processes the HTTP request by creating the following objects:

- javax.servlet.ServletContext
- javax.servlet.HttpServletRequest
- javax.servlet.HttpServletResponse
- javax.servlet.http.HttpSession

Conversely the second time an HTTP request comes for the same Servlet/JSP from the same browser, the container creates only the following objects:

- javax.servlet.HttpServletRequest
- javax.servlet.HttpServletResponse

The second time an HTTP request comes for the same Servlet/JSP but from a different browser, the container creates only the following objects:

- javax.servlet.HttpServletRequest
- javax.servlet.HttpServletResponse
- javax.servlet.http.HttpSession

The Servlet/JSP container creates one javax.servlet.ServletContext object per Web application. This object provides application-wide services for all Servlet/JSPs within that Web application. The Servlet/JSP container creates one javax.servlet.HttpSession object for each browser connection. Each HttpSession object provides services to the corresponding browser

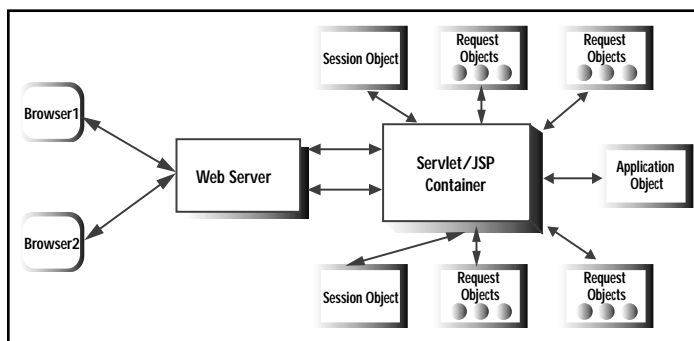


FIGURE 1 The architecture of the Servlet/JSP container. For each Client, a session object is created, and for each HTTP request, a request and response object is created. Each HTTP request is handled in a separate thread within the Servlet/JSP container.

connection and is responsible for maintaining the individual browser information. Each time an HTTP request comes from a browser that contains a Servlet/JSP, the container creates new javax.servlet.HttpServletRequest and javax.servlet.HttpServletResponse objects. The HttpServletRequest provides browser request information to a Servlet/JSP. It also provides data, including parameter names and values, that's set in the browser. The Servlet/JSP uses the HttpServletResponse object to send responses back to the browser.

Another important concept of the Servlet/JSP container is that when multiple HTTP requests from different browsers come to a Servlet/JSP, the container handles it by creating a thread for each request. This significantly increases the performance of the Web application. However, this feature also creates a problem that will be addressed in the next section.

Life Cycle of the Objects

In the previous section we discussed the Servlet/JSP container's runtime environment. In this section we'll look at the life cycle of the HttpSession, HttpServletRequest, and HttpServletResponse objects that make the runtime environment. The ServletContext object exists as long as the Web application is active in the Servlet/JSP container. When the Web application is either closed or restarted, the corresponding ServletContext object becomes a potential candidate for garbage collection by the Java virtual machine.

The HttpSession object that's created for each browser connection remains within the Servlet/JSP container as long as the session is active. When the session is inactive for a specific time period, the Servlet/JSP container removes that object and makes it a potential candidate for garbage collection.

When the Servlet/JSP container receives a new HTTP request from the browser, it removes the references to the old HttpServletRequest and HttpServletResponse objects and creates new ones.

The HTTP is a stateless protocol, that is, it doesn't maintain the state of the HTTP requests from the browser. Each time a new HTTP request comes from the browser, it doesn't maintain the state (session data) of the previous HTTP request. In most Web applications it's always necessary to maintain the user session so that content is sent to the correct browser. If not, the user must log in each and every time, not an acceptable solution. By creating the HttpSession, HttpServletRequest, and HttpServletResponse objects at the appropriate times, the Servlet/JSP container provides a solution to the above problem. However, this solution is primitive and has its own problems. We need to extend this concept so we can have a scalable, maintainable, and adaptable architecture.

Session Data Storage

Session data can be stored at the client- or server-side. In the client-side approach we can use the following techniques: cookies, URL rewriting, and hidden fields. In the server-side approach we can store the session data in a persistence store that can be retrieved for later use.

Client-Side Cookies

We can use client-side cookies to store all the session data with the client ID. Once the session data has been written, it can be retrieved later using this ID. By converting all the session data into a string, it's possible to store an entire session data into a cookie. Listing 1 shows how you can use, store, and retrieve cookies.

Once the cookie has been written, you can retrieve it from an HttpServletResponse object using getCookie(). Cookies are easy to implement but they have several limitations: a cookie can only store up to 4K of text, and there are restrictions on the number of cookies that can be stored in a given domain.

Client-Side HTML Hidden Fields

Another approach to preserving session data is using "hidden fields," available in HTML INPUT tags. The stored session data can be retrieved using the HttpServletRequest class that's defined by the getParameter() method. Although this approach is easy to use and implement, it has its

own problems – we can store only string types to the hidden fields, and we can't store other objects. Another major problem with this approach is that the performance of the Web site decreases significantly. First, a request comes from a browser, you process that request, store the result in the hidden fields, and send that HTML page back to the browser for additional user requests. Next, a request comes from the same user, you process the first request as well as the second one, and send the result back to HTML. With this approach you end up processing the same data again and again so it's not suitable for large Web sites.

Client-Side URL Rewriting

Generally this technique is used when the user disables the cookies in the browser. You append a client ID as a response parameter to all URLs that are served by the Web server. This is accomplished by using the `encodeURL()` method defined in the `HttpServletResponse` class. One problem with this approach is that every URL on every page must be rewritten dynamically in order to embed the client ID in every request. Listing 2 provides an example of URL rewriting. In it the `itemID` is used as a client ID that identifies the user in case the browser is disabled by the cookies. *Note:* You need to generate these URLs for each and every link in all your pages, which is tedious and difficult to maintain.

Another problem with this approach is *security*. If we store all the session data in the client, with little effort anyone can access that data. One way to solve the problem is to encrypt the session data before storing it on the client-side. However, even if you encrypt the session data it's still not a good practice to store sensitive business data at the client-side.

Server-Side Persistence Approach

In this approach we use the client-side to store only the client ID (using cookies or hidden fields); the session data associated with this ID is stored in the server. When the cookies are turned off in the browser, we can use URL rewriting to send the client ID to the browser. Using the method `encodeURL()` that's defined in the `HttpServletResponse` class solves all the problems mentioned earlier in the client-side approach.

Problems with the Servlet/JSP Container Programming Model

In this section we'll explore the inherent problems associated with the Servlet/JSP programming model. As explained earlier, it's a complex task when multiple HTTP requests come for the same Servlet/JSP. The Servlet/JSP container creates a new thread to handle each request. So all the threads share the same instance variables in the Servlet/JSP. This makes it impossible to store the different client information in the instance variables in the Servlet/JSP. The only option is to use the `HttpSession` object to store the instance variables, which can be retrieved or removed as necessary. But adding and removing instance variables to the `HttpSession` object from any Servlet/JSP will create unmanageable spaghetti code. Similarly, accessing the `HttpRequest` and `HttpResponse` objects directly from the Servlet/JSP will also create unmanageable code. So we need a mechanism that helps us use the `HttpSession`, `HttpServletRequest`, and `HttpResponse` objects in a more controlled and manageable way.

When developing Web applications, map the browser requests to the appropriate Servlet/JSP. The easiest way to map the browser actions to the back-end Servlet/JSP is to use a query parameter within the HTML forms or URL links and have an if-then-else statement to find the appropriate Servlet/JSP. Although this approach is simple, it creates a perfor-

Listing 1

```
//Setting Cookies
package test;
import javax.servlet.*;
public class CookieSetter(HttpServletResponse resp, String
sessionData){
    //Client Id is "CybelinkCookie"
    Cookie coo=new Cookie("CybelinkCookie", sessionData);
    coo.setDomain("cybelink.com");
    coo.setPath("/");
    coo.setMaxAge(24*60*60) //One day
    coo.setVersion(0);
    resp.addCookie(coo);
}

//Retrieving Cookies
package test;
import javax.servlet.*;
public class CookieRetriever(HttpServletRequest req, String
sessionData){
    Cookie[] cookies=req.getCookies();
    Cookie found=null
    if(cookies != null){
        for(int i=0; i<cookies.length; ++i){
            if(cookies[i].getName.equals("CybelinkCookie")){
                found=cookies[i];
                break;
            }
        }
    }
}
}
```

Listing 2

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

import java.io.*;

public class ItemsCatalogServlet extends HttpServlet {

    public void doGet (HttpServletRequest req,
        HttpServletResponse resp)
        throws ServletException, IOException{
        //....get ShoppingCart
        Items[] items = shoppingCart.getItems();
        for(int i=0; i < items.length; i++) {
            out.println("<a href=\""
+response.encodeURL("/servlet/items?bookId=" +bookId)+
                "\"> <strong>" + items[i].getName() +
            "</strong></a></td>");
        }
    }
}
```



mance nightmare for Servlet/JSP developers. If we have to support new Servlet/JSPs, we need to change the source code; this leads us to the development life cycle of compile, test, debug, and deploy.

Another significant problem with developing Web applications is managing the changes in Web resources such as HTML, Servlet, and JSP. During development and production time, these resources change names, directories, paths, and more. If we hard-code these names and paths in our programs, for each and every change we have to go through the compile, debug, test, and deploy cycles. An alternative is to store the names and paths in a text file and access them from the programs. If there's any change in the Web resources, we can edit the text file and simply restart the Web application.

High-Traffic Web Sites

The standard JSDK implementation stores the HttpSession object in the memory of the JVM where it was created so an HttpSession can be retrieved efficiently. Though this approach works for smaller Web sites or sites that don't need session information from one request to another, this approach doesn't scale well in larger, high-traffic e-business sites.

In those sites where multiple Web servers are used to serve Web pages, a load balancer (either hardware or software) is used to distribute the HTTP traffic to different Web servers. When the load balancer assigns a Web server to a particular HTTP request that comes from a browser, all subsequent requests from that browser are assigned to the same Web server. The load balancer accomplishes this by examining the IP address of the incoming packets and assigning a particular Web server to that packet, as well as assigning the same Web server to all the packets that have the same IP address. This method of assigning the packets with the same IP address to the same Web server is called *server affinity*. Server affinity may not be guaranteed because many companies now assign random IP addresses to the outgoing packets. In this case the load balancer can't assign the same Web server to packets that come from the same browser. If we use the reference implementation of JSDK, we'll run

into problems maintaining the HttpSession objects. The reference implementation of JSDK maintains the HttpSession object in the memory of the JVM in which it was created. When the load balancer assigns the packets from the same browser to different Web servers, the HttpSession object created for the first packet is lost to subsequent packets. To avoid this problem we need to store the HttpSession object in the persistence store, which must be accessible to all Web servers, so the right HttpSession object can be retrieved using the session ID stored in the cookie.

Summary

In Part 1 we described the inherent problems associated with J2EE's Servlet/JSP container and how they affect the creation of enterprise-wide, scalable Web architecture. In Part 2, we'll look at Jlink's architecture and how it solves these problems.

References

1. Sun's J2EE Blueprint: <http://java.sun.com/j2ee/blueprints>
2. Buschmann, F., et al. (1996). *Pattern-Oriented Software Architecture*. John Wiley & Sons.
3. Fields, D.K., and Kolb, M.A. (2000). *Web Development with JavaServer Pages*. Manning Publications.
4. *Servlet/JSP API*: <http://java.sun.com/products/servlet/2.2/javadoc/index.html> 🍌

AUTHOR BIO

Mani Malarvannan is CEO and cofounder of Cybelink (www.cybelink.com), a Minnesota-based company that specializes in e-business consulting and training. Mani has several years of OO analysis and design experience and has been working in Internet and Java technologies since their inception. He holds BS and MS degrees in computer science.

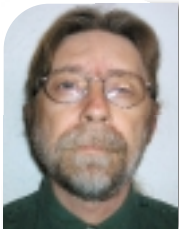
mani@cybelink.com

Consolidating Legacy Data

New ways to access back-end systems

Part 2 of 2

WRITTEN BY
BRADY FLOWERS



In Part 1 of this article (*JDJ*, Vol. 6, issue 1) we discussed solving legacy data integration problems with VisualAge for Java and WebSphere Studio. In Part 2 we'll discuss using the MQSeries Integrator and some of the steps for creating data translations and data flows.

MQSeries Integrator extends MQSeries by adding message brokering that's driven by business rules. MQSI lets us add the intelligence to route and transform messages or filter messages (content-based or topic-based). It also lets us perform direct

database access so we can augment or warehouse messages. We'll look primarily at routing and transforming as we build our solution.

The MQSI Control Center lies at the heart of MQSI's user interface. We'll use this tool to create message types and message flows. Figure 1 shows an example message flow, one that would be applicable to our problem. Notice the similarity between this screen and the VisualAge for Java Visual Composition Editor. Components are dropped on the canvas and wired together in much the same manner as Java objects are in VisualAge.

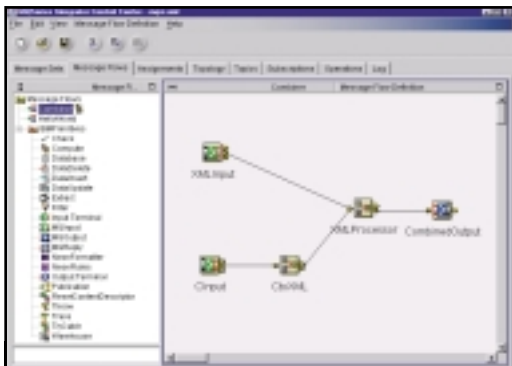


FIGURE 1 Proposed message flow

The Message Flow

In our sample message flow we left out any error or exception handling so we could highlight the core tasks. As you can see from Figure 1, the message flow describes our stated problem: it takes two disparate inputs and merges them into a single XML output stream. Both the C-formatted data and the XML data from our legacy applications arrive on message queues – the MQInput nodes. The C input is routed through a processor, or “compute node,” that will translate the data into XML. Both streams are then sent off to an XML processor compute node, where they're merged and placed on an output queue.

Notice that we haven't defined how the data gets onto the input queues in the first place. In the complete solution, we'll define another message flow that sends request messages to both legacy applications with the appropriate search criteria for locating a customer. The input mes-

sages for the “Combiner” flow represent the replies from such a query. Different queries could be posed – a search by name or postal code, for example. The message flow defined here could be used to process the results of any such query.

Before we get too deeply into the message flow, we need to use the MQSI Control Center's Message Repository Manager to define the message that will encapsulate the C-formatted data. There are three steps needed to create a Message Repository Manager (MRM) message from a C structure:

1. Create the message set.
2. Import a C structure to create a new message type within the message set.
3. Create a message of that new message type.

The creation of the message set only involves choosing a menu option on the Message Sets pane and giving the new message set a name. MQSI can read a C header file or a COBOL copybook, parse the contents, and generate a message type based on the structures or COMMAREAS defined in the file. We've used the C header file from Listing 1 (reproduced from Part 1 of this article) to create a completed message type with a minimal amount of typing and clicking on our part.

Now we can return to our message flow. Each of the input nodes has properties in which we can enter the queue name that will be used and the message format that will be presented. For the CInput node, the message domain will be MRM and the format will be our generated type, C_CUSTOMER_TYPE. For the XMLInput node, the domain is XML.

```

C_MESSAGE
  Message Set: C_Message
  Input Queue:
  Output Queue:
  Message Format:
  Message Domain:

  [ ] Copy message headers only [ ] Copy entire message

  Message Flow:
  DECLARE 3 SPTRN;
  SET I = 1;
  WHILE I = CASHCMTN(Exporters+1) DO
    SET OutputQueue(1) = InputQueue(1);
    SET I=I+1;
  END WHILE;

  SET InputQueue(1) = InputQueue(1);
  SET OutputQueue(1) = InputQueue(1);
  SET OutputQueue(2) = InputQueue(2);
  SET OutputQueue(3) = InputQueue(3);
  SET OutputQueue(4) = InputQueue(4);
  SET OutputQueue(5) = InputQueue(5);
  SET OutputQueue(6) = InputQueue(6);
  SET OutputQueue(7) = InputQueue(7);
  SET OutputQueue(8) = InputQueue(8);
  SET OutputQueue(9) = InputQueue(9);
  SET OutputQueue(10) = InputQueue(10);
  SET OutputQueue(11) = InputQueue(11);
  SET OutputQueue(12) = InputQueue(12);
  SET OutputQueue(13) = InputQueue(13);
  SET OutputQueue(14) = InputQueue(14);
  SET OutputQueue(15) = InputQueue(15);
  SET OutputQueue(16) = InputQueue(16);
  SET OutputQueue(17) = InputQueue(17);
  SET OutputQueue(18) = InputQueue(18);
  SET OutputQueue(19) = InputQueue(19);
  SET OutputQueue(20) = InputQueue(20);
  SET OutputQueue(21) = InputQueue(21);
  SET OutputQueue(22) = InputQueue(22);
  SET OutputQueue(23) = InputQueue(23);
  SET OutputQueue(24) = InputQueue(24);
  SET OutputQueue(25) = InputQueue(25);
  SET OutputQueue(26) = InputQueue(26);
  SET OutputQueue(27) = InputQueue(27);
  SET OutputQueue(28) = InputQueue(28);
  SET OutputQueue(29) = InputQueue(29);
  SET OutputQueue(30) = InputQueue(30);
  SET OutputQueue(31) = InputQueue(31);
  SET OutputQueue(32) = InputQueue(32);
  SET OutputQueue(33) = InputQueue(33);
  SET OutputQueue(34) = InputQueue(34);
  SET OutputQueue(35) = InputQueue(35);
  SET OutputQueue(36) = InputQueue(36);
  SET OutputQueue(37) = InputQueue(37);
  SET OutputQueue(38) = InputQueue(38);
  SET OutputQueue(39) = InputQueue(39);
  SET OutputQueue(40) = InputQueue(40);
  SET OutputQueue(41) = InputQueue(41);
  SET OutputQueue(42) = InputQueue(42);
  SET OutputQueue(43) = InputQueue(43);
  SET OutputQueue(44) = InputQueue(44);
  SET OutputQueue(45) = InputQueue(45);
  SET OutputQueue(46) = InputQueue(46);
  SET OutputQueue(47) = InputQueue(47);
  SET OutputQueue(48) = InputQueue(48);
  SET OutputQueue(49) = InputQueue(49);
  SET OutputQueue(50) = InputQueue(50);
  SET OutputQueue(51) = InputQueue(51);
  SET OutputQueue(52) = InputQueue(52);
  SET OutputQueue(53) = InputQueue(53);
  SET OutputQueue(54) = InputQueue(54);
  SET OutputQueue(55) = InputQueue(55);
  SET OutputQueue(56) = InputQueue(56);
  SET OutputQueue(57) = InputQueue(57);
  SET OutputQueue(58) = InputQueue(58);
  SET OutputQueue(59) = InputQueue(59);
  SET OutputQueue(60) = InputQueue(60);
  SET OutputQueue(61) = InputQueue(61);
  SET OutputQueue(62) = InputQueue(62);
  SET OutputQueue(63) = InputQueue(63);
  SET OutputQueue(64) = InputQueue(64);
  SET OutputQueue(65) = InputQueue(65);
  SET OutputQueue(66) = InputQueue(66);
  SET OutputQueue(67) = InputQueue(67);
  SET OutputQueue(68) = InputQueue(68);
  SET OutputQueue(69) = InputQueue(69);
  SET OutputQueue(70) = InputQueue(70);
  SET OutputQueue(71) = InputQueue(71);
  SET OutputQueue(72) = InputQueue(72);
  SET OutputQueue(73) = InputQueue(73);
  SET OutputQueue(74) = InputQueue(74);
  SET OutputQueue(75) = InputQueue(75);
  SET OutputQueue(76) = InputQueue(76);
  SET OutputQueue(77) = InputQueue(77);
  SET OutputQueue(78) = InputQueue(78);
  SET OutputQueue(79) = InputQueue(79);
  SET OutputQueue(80) = InputQueue(80);
  SET OutputQueue(81) = InputQueue(81);
  SET OutputQueue(82) = InputQueue(82);
  SET OutputQueue(83) = InputQueue(83);
  SET OutputQueue(84) = InputQueue(84);
  SET OutputQueue(85) = InputQueue(85);
  SET OutputQueue(86) = InputQueue(86);
  SET OutputQueue(87) = InputQueue(87);
  SET OutputQueue(88) = InputQueue(88);
  SET OutputQueue(89) = InputQueue(89);
  SET OutputQueue(90) = InputQueue(90);
  SET OutputQueue(91) = InputQueue(91);
  SET OutputQueue(92) = InputQueue(92);
  SET OutputQueue(93) = InputQueue(93);
  SET OutputQueue(94) = InputQueue(94);
  SET OutputQueue(95) = InputQueue(95);
  SET OutputQueue(96) = InputQueue(96);
  SET OutputQueue(97) = InputQueue(97);
  SET OutputQueue(98) = InputQueue(98);
  SET OutputQueue(99) = InputQueue(99);
  SET OutputQueue(100) = InputQueue(100);
  
```

FIGURE 2 C to XML translation

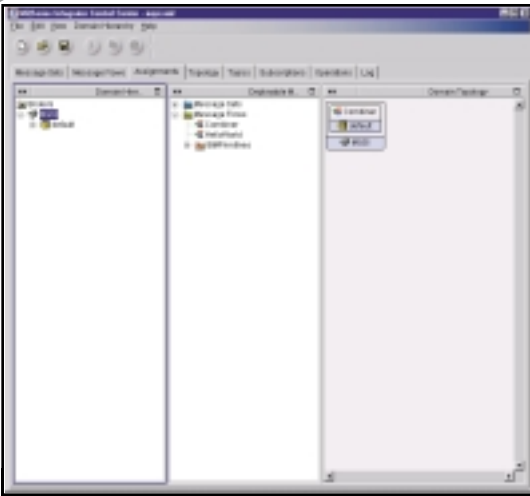


FIGURE 3 Deployed message flow

Notice how we've structured the flow so the C structure is translated to XML before further processing. Figure 2 shows the completed property sheet for the CtoXML Compute node. We added our MRM message type C_CUSTOMER_TYPE and wrote the ESQL for the translation. To convert the C structure to XML, we included the line:

```
SET OutputRoot.Properties.MessageFormat = 'XML';
```

We then used OutputRoot.XML in the lines that follow to set the field values in the output XML. Notice that we changed the field names between input and output and that we didn't include the fields that aren't required by our Web application.

Because we chose to perform this translation before sending both streams to a common processor, the XMLProcessor can concern itself with the job of merging two XML streams without knowing that one or more of its inputs started life in some other format. Again, here's a good opportunity to create a reusable component.

Finally, we can use the MQSI Control Center to deploy our message flow to the running broker. We can do this from the Assignments page. Figure 3 shows a completed deployment.

Java Access to MQSeries

We have several ways to access MQSeries from the code we'll develop in VisualAge for Java. MQSeries has featured a robust Java API for several years. The original MQSeries Java API predates the Java Message Service (JMS) API but since the introduction of JMS, IBM also provides a JMS implementation for MQSeries so that developers can write portable code to access MQSeries. Final-

ly, the IBM Common Connector Framework (CCF) features an MQ connector. CCF is quite powerful and flexible, and it provides the basis for the upcoming Java Connector API in J2EE.

We took a middle-of-the-road approach here and created a wrapper class called MQAccess that we'll use for our queue access. Right now this class is implemented using the MQSeries "original flavor" API, but we could change the implementation at any time without affecting our business logic. This class is shown in Listing 2. It uses the IBM XML4J Parser for part of its data handling, so that feature must be added to your VisualAge for Java workspace.

We've provided a main() method to illustrate how the methods in the class should be called. To run this class, however, there are other steps that still need to be taken, not the least of which is to set up MQSeries, create something to mimic the legacy systems during testing, and start the MQSI Broker. This all falls outside the scope of this article but you can get a complete solution from the "Patterns for e-business" kit that's available from IBM.

Finishing Off

Once the rest of the infrastructure is set up and our MQAccess class is tested in the VisualAge for Java environment, the next step is to move the class into

WebSphere Studio so we can create the wrapper servlet, HTML, and JSP for the Web application. From this point we can deploy to test and production servers or to the VisualAge for Java WebSphere Test Environment; move Java code back and forth between Studio and VisualAge for editing, testing, and debugging; and further enhance the HTML and JSP files. For a discussion on how to do all these things, see *JDJ* (Vol. 5, issues 9, 10).

Summary

Hopefully, in this brief discussion of MQSeries Integrator we've been able to show how valuable a message broker and translation/routing engine based on business rules can be when you need to solve a problem that involves the integration of legacy data, regardless of the format and the location of the data or legacy systems. Once we created our message flows, we created a flexible set of tools – reusable components we can build upon as we need to access back-end systems in new ways. The MQAccess class we created could serve as the basis for a set of EJB components. It could service stateless session beans for generic MQSeries access or even be the engine for BMP entity beans. ☺

bradenf@us.ibm.com

Listing 1

```
#define CUSTNO_LEN 8
#define FNAME_LEN 24
#define LNAME_LEN 24
#define ADDR_LEN 24
#define CITY_LEN 24
#define STATE_LEN 2
#define ZIP_LEN 10

struct C_CUSTOMER {
    char custno[CUSTNO_LEN];
    char fname[FNAME_LEN];
    char lname[LNAME_LEN];
    char addr[ADDR_LEN];
    char city[CITY_LEN];
    char state[STATE_LEN];
    char zip[ZIP_LEN];
    double balancedue;
    int datedue_month;
    int datedue_day;
    int datedue_year;
}
```

Listing 2

```
public class MQAccess {
    private java.lang.String hostname = "localhost";
    private java.lang.String channel = "JAVA.CHANNEL";
    private java.lang.String userid = null;
    private java.lang.String password = null;
    private java.lang.String qMan-
```

```
agerName = null;
    private com.ibm.mq.MQQueueManager qManager = null;
    private int defaultMessageSize = 100;

    public MQAccess() {
        super();
    }

    public void connectQManager()
        throws com.ibm.mq.MQException {
        disconnectQManager();
        setEnvironment();
        qManager = new
            com.ibm.mq.MQQueueManager(qManagerName);
    }

    public void connectQManager(String
        newQManagerName) throws
        com.ibm.mq.MQException {
        setQManagerName(newQManagerName);
        connectQManager();
    }

    public void disconnectQManager()
```

AUTHOR BIO

Brady Flowers is a software IT architect with IBM's WebSpeed team specializing in WebSphere, Java, and the rest of IBM's suite of e-business applications.



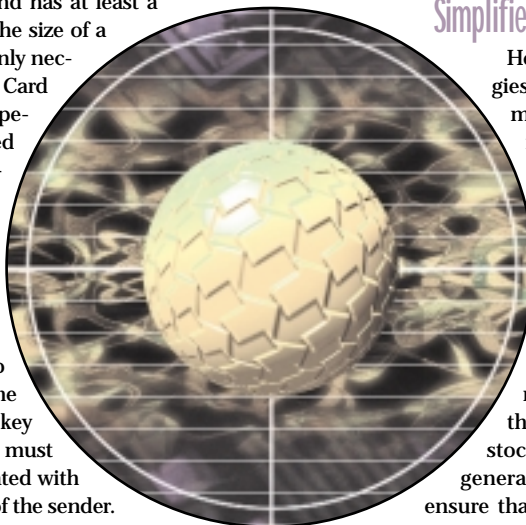
Using Motorola's Java Card to Digitally Sign a Message

Integrating Java, smart cards, & cryptography

WRITTEN BY ANDREW WEBB

Smart card, Java workstation, and cryptography: these are all growing areas of interest in the computing world. There are programmers – from novice to expert – who know each of these technologies. But as the technology world becomes more intertwined, so too do these seemingly disparate technologies. With the introduction of the Java Card it's becoming necessary for smart card developers to know Java and Java developers to know smart cards. And as more transactions are done electronically, everyone will try to find ways of applying cryptography to electronic security. The purpose of this article is to provide a sample solution that brings all three of these technologies together.

Assuming the reader knows what Java is and has at least a vague notion of what a smart card is (a card the size of a credit card containing a microprocessor), it's only necessary to bring the reader up to speed on a Java Card and cryptography. Basically, a Java Card is a specialized type of smart card that's programmed using portable Java applets instead of proprietary commands. Cryptography can be used to digitally sign a message that can be checked, but not created, by those with access to the public key of the sender. The cryptographic algorithm required for this purpose uses a public and private key pair that's created concurrently. The sender uses the private key to encrypt a message, and the receiver uses the public key to decrypt the message. If the public key successfully decrypts the message, the sender must be the proper owner of the private key associated with that public key, therefore proving the identity of the sender.



Simplified High-Level Example

Here's an example of how these three technologies could be combined. Suppose I want to send a message to my stockbroker telling him to sell all my shares of Microsoft. Now my stockbroker wants to make sure it's really me sending the message. So using his Java workstation and a couple of extra cryptography applications, he generates a public/private key pair. He stores the private key on the Java Card and issues it to me. When I want to send a message to him, I use an applet on the Java Card. It digitally signs the message with the private key and returns a signature to my workstation. I can then e-mail the message with the signature to my stockbroker, who knows the public key (because he generated it). He can then decrypt the signature, ensure that I had sent the message, and sell my shares.

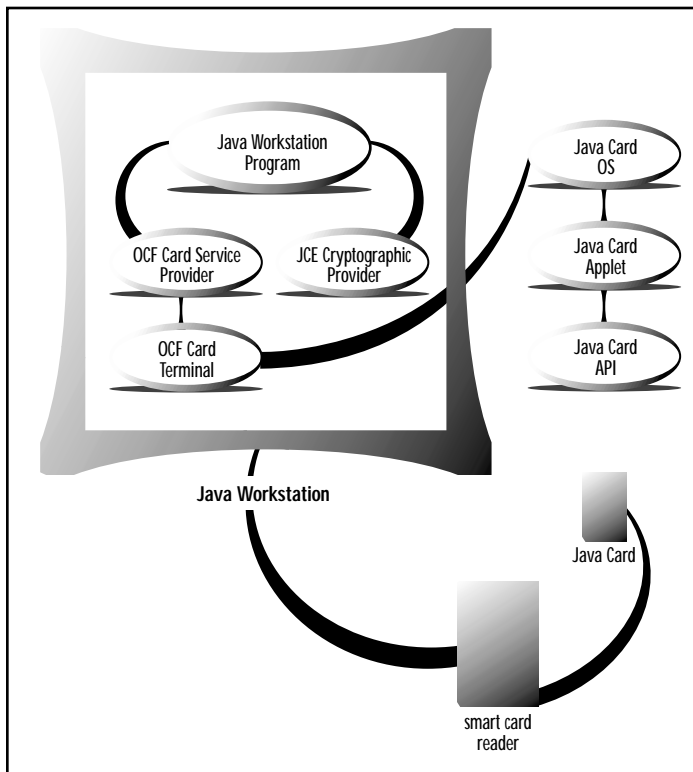


FIGURE 1 Software and hardware layers

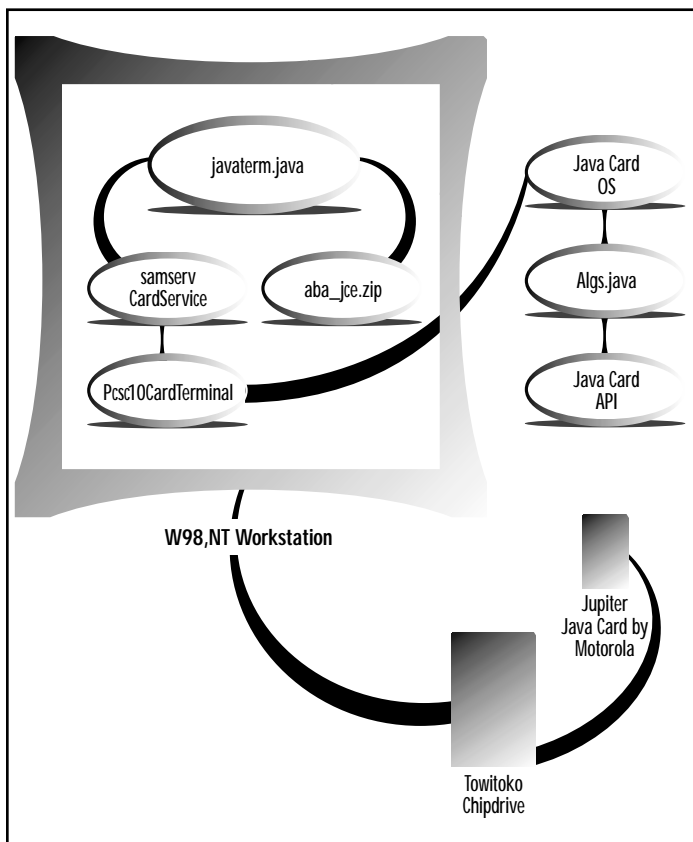


FIGURE 2 Development environment used in the sample solution

Note that the public key can verify the signature, but only the private key can generate it. The advantage of this solution is that anyone with access to the public key can verify the signature, and the public key doesn't have to be kept secret. The private key, however, remains on a secure, portable token – the Java Card.

Goal

The remainder of this article describes how to integrate the various Java-based tools to use public key technology on a Java Card. Included in the sample solution are the steps to generate an asymmetric key pair, load it onto a Java Card, sign a message with a private key on the card, and verify the signature using the public key. The article describes the design of the Java workstation program, Open Card Framework (OCF) Card Service Provider, and the Java Card applet on the Java Card. All other components are off-the-shelf applications that are available on the Internet.

Description of Development Environment and Hardware/Software Layers

Before starting, it's necessary to describe the programming environment used in this sample solution and understand how the layers fit together. Figure 1 shows, in general terms, the software and hardware layers.

Referring to Figure 1, the following is a brief description of how the pieces of the development environment work together.

The Java workstation program calls the JCE Cryptographic Provider to generate the RSA key pair. The Java workstation program then calls the OCF Card Service Provider that generates a card command to load the private key. The card command is sent from the OCF Card Service Provider to the OCF CardTerminal. Note that the OCF Card Service Provider is responsible for card-specific operations, while the OCF CardTerminal deals only with commands specific to the smart card reader.

The OCF CardTerminal driver tells the physical reader to send the command to the card. The Java Card OS passes the command to the appropriate applet on the card. The applet handles the command, usually passing data back, calling the Java Card 2.1 API, or changing the state of the applet.

Data returned from the applet travels from the Java Card OS back to the smart card reader, then to the OCF CardTerminal, and finally to the Card Service Provider.

The OCF Card Service Provider generates card commands that are interpreted by the Java Card applet. The definition of the commands is a shared source between the applet and the Card Service Provider.

More specifically, Figure 2 shows the exact software and hardware development environment used in creating this solution. The actual development environment is important to know should the reader want to attempt the same solution. Some steps may differ depending on the environment used.

The Motorola M-Smart Jade Workbench handles the generation of a CAP file and loads it onto a Motorola Java Card using Visa Open Platform (VOP) 2.0 commands. It can also be used to simulate and test a card applet without using a physical card and reader. It's a commercial product that comes with sample Java Cards and a reader.

The Solution

Now that the relevant background information has been provided, it's time to get down to the actual steps of implementing the solution. These steps are broken down into two main parts:

1. Programming for the Java workstation
2. Programming for the Java Card

Java Program on the Workstation or PC

Step 1: Design of the Java workstation program

It's important to first understand that while the Java Developer's Kit 1.2.2 will support the code for calls made to cryptography routines, it currently doesn't have the cryptography functionality built in. Specifically, the Java Cryptography Extension (JCE) has the structure defined to generate an RSA key pair, and there are Open JCE Java

Cryptography Extension Provider implementations available from third parties.

The code to generate a key is:

```
KeyPairGenerator keyGen = KeyPairGenerator.getInstance("RSA");
keyGen.initialize( keylen, random);
KeyPair pair = keyGen.generateKeyPair();
```

In this code, keylen is perhaps 512 bits, and random comes from these steps:

```
SecureRandom random = SecureRandom.getInstance("SHA1PRNG", "SUN");
random.setSeed( 1234 );
```

(Note: Since the seed is set from a constant, this will produce the same key pair each time. This is helpful for testing, but in the real application you'd like to come up with an indeterminate method for initializing the seed.)

To send the key to the card, it must be broken down into its various components, which can be done with this code:

```
PrivateKey priv = pair.getPrivate()
byte [] enc_key = priv.getEncoded();
```

enc_key is a byte array that now contains the various key components in PKCS-8 format. The modulus and private exponent that make up the private RSA key can be extracted by using Appendix A as an example. (All appendices in this article can be found at www.java-developers.com.) The private key is sent to the card using the "samservCardService" class, and the private key is no longer needed on the workstation. The code for this can be found in the file samservCardService.java, which is available on the Motorola Smart Card Web site, www.motorola.com/smartcard/.

Now we have a message, "I will pay Fred 8 dm," that we want to sign with the private key. We simply send the message in a signing command as shown below and receive the result.

```
ss.Algs( samservCardService.SIGN_OPER, samservCard-
Service.RSA512PRIV_ALG, (byte)20, 0, data );
```

The complete code for this can be found in the file javaterm.java on the **JDJ** Web site.

Step 2: Connecting to the Java Card from the Java workstation program

This step focuses on the smart card connectivity. The code below shows how to start the OCF CardTerminal, initialize the samservCardService instance, send an application program data unit (APDU – a message sent to a smart card that tells the card to do something), and then close smart card operations.

```
SmartCard.start();

CardRequest cr = new CardRequest (CardRequest.ANY-
CARD, null, samservCardService.class );
SmartCard sm = SmartCard.waitForCard( cr );

samservCardService ss = (samservCardService) sm.get-
CardService( samservCardService.class, true);

ss.SelectApplet(); // Make our Javacard applet run.
```

```
SmartCard.shutdown ();
```

This is a simple version, and more details can be found in the *Open-Card Framework 1.2 Programmer's Guide* included with OCF.

Step 3: Design of the OCF Card Service Provider code

Two application program data units define the samservCardService: (1) select algsapp and (2) AlgsOperate. The operations associated with the AlgsOperate APDU include Load Key (modulus), Load Key (exponent), Encrypt, Decrypt, Sign, Verify, and Show Key.

The Java Card applet on the card has to support those two commands, while the samservCardService simply generates the commands and sends them to the OCF CardTerminal using the member function sendCommandAPDU(). A CardChannel must be obtained from the inherited member functions allocateCardChannel(), getCardChannel(), and releaseCardChannel().

The code below shows how a member function of a class that extends

"CardService" can send an APDU to a card via an OCF driver.

```
public void SelectApplet() /* Select the
"algsapp" Javacard applet */
throws CardTerminalException
{
    byte[] SelAid= { (byte)0x00, (byte)0xa4,
(byte)0x04, (byte)0x00, (byte)0x07,
    (byte)0x61, (byte)0x6c, (byte)0x67,
(byte)0x73, (byte)0x61, (byte)0x70, (byte)0x70
};

    CommandAPDU capdu = new CommandAPDU( SelAid,
12 );
    try {
        allocateCardChannel();
        rapdu = getCardChannel().sendCommandAPDU
( capdu );
    } finally {
        releaseCardChannel();
    }
}
```

Step 4: Checking the signature with the public key

Now that the message has been signed using the private key, it can be verified by decrypting the signature with the public key. Before doing this, it's necessary to jump ahead and understand what was done on the card. This is explained in detail in Step 1 of the Java Card Applet found on the **JDJ** Web site. In the meantime Figure 3 shows what happens as the message (or data) is sent to the Java Card and the signature is generated.

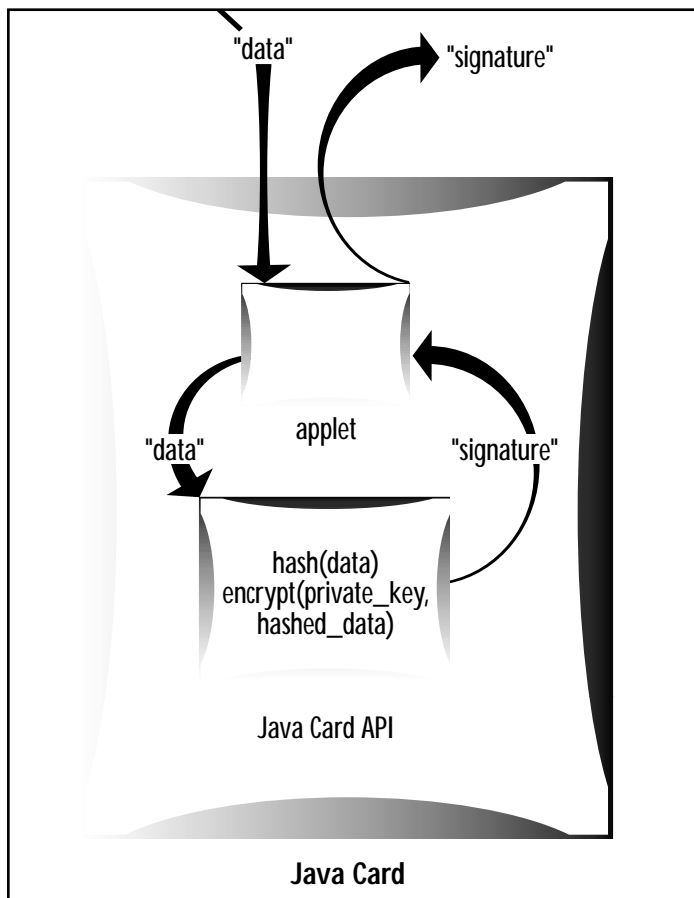


FIGURE 3 How the signature generation works

“
With the introduction of the Java Card it's becoming necessary for smart card developers to know Java and Java developers to know smart cards
”

The Java applet on the card automatically hashes the message before encrypting it and returning the signature.

```
Cipher rsa_ciph =
Cipher.getInstance("RSA/ECB/PKCS1Padding");
rsa_ciph.init( Cipher.DECRYPT_MODE, pair.getPublic() );
rsa_ciph.doFinal( sig_of_data, sig_off, sig_len,
de_sig );
```

The code above decrypts the signature, and the result is the SHA-1 hash of the original data in the byte array de_sig. Since we can't "unhash" this to verify that it matches the original message, we can hash the original data ourselves. If the SHA-1 hash of the original data matches the decrypted hash returned by the Java Card, we've verified that the public key is the one associated with the private key. The code below shows how to generate a SHA hash of the original data. If the signature is correct, the hash in de_sig and the recomputed hash should be equal. The hash is either in the first 20 bytes of de_sig or directly after a 15-byte SHA-1 identifier.

```
MessageDigest sha =
MessageDigest.getInstance("SHA");
sha.update( data );
byte[] hash = sha.digest();
```

Step 5: Building and running the Java workstation program using the Sun JDK

The following files, available as part of the OCF installation, must be in your Java classpath: base-core.jar, base-opt.jar, jce1_2-do.jar, and jce.zip

When using the OCF-provided PC/SC driver, you also need \OpenCard\OCF1.2\lib in the OS path to pick up the OCF pc/sc dlls, and you must reference terminals-windows.jar and pcsc-wrapper.jar in the Java classpath. (Naturally, you also need a functional PC/SC installation.)

Assuming that CPATH contains all the relevant JARs and zips, the JDK commands in the listing below will build the Java class file and combine the CardService and its factory into a JAR file.

```
javac -classpath %CPATH% javaterm.java
cd samserv
javac -classpath %CPATH% samservCardService.java samservCardServiceFactory.java
cd ..
jar cvf samserv.jar samserv\samservCardService.class samserv\samservCardServiceFactory.class
```

The path environment variable includes C:\;JDK1.2.2\BIN;C:\OPENCARD\OCF1.2\LIB. The command java -cp %CPATH% javaterm will then produce the output shown in Appendix B.

Java Card Applet

Now that the steps from the Java workstation side have been explained, it's time to take a look at what's happening at the card level.

Step 1: Design of the Java Card 2.1 applet

The details of the Java Card 2.1 runtime environment are covered in Sun's Java Card 2.1.1 Runtime Environment (JCRE) Specification. Briefly though, the algs class extends the applet and handles the input and output of the APDU. After an APDU has arrived in the buffer the Java Card environment calls the process (APDU apdu) function. The member

variables of algs are held in persistent storage, while local variables use the stack.

Under the constructor Algs(), the cryptographic member variables are created with their defining parameters, such as key size, and the type and hashing for the signature. Naturally, they have to be in agreement with the parameters used on the terminal side.

The process() method has a switch that contains the essential cryptographic manipulations of loading the private key and using it to encrypt and sign data. In all cases we use the data from the incoming command as it's sent to the card.

Step 2: Building the Java Card applet with the Sun JDK, using a jar file from Motorola

The commands shown below are those needed to build a JAR file from the algs.java file.

```
javac -g -classpath "\program files\M-Smart JADE\jcapi.jar"
algs.java
cd ..
jar cvf algs/algs.jar algs/algs.class
cd algs
```

To convert the JAR file into a CAP file, which is the file used to load the applet into the card, use M-Smart Jade Workbench. A CAP file is specified in Sun's jcvmspec.pdf, section 6.

Notice that the jcapi.jar (Java Card API) came from the installation directory of the Motorola M-Smart Jade Workbench.

Step 3: Generating and loading the CAP file using the Motorola M-Smart Jade Workbench

To build the .CAP, you start Jade, and select Card | Jupiter.

Note: You can't run Jade on a Windows PC if PC/SC is locking the serial port. The symptom is that Jupiter doesn't appear below simulator on the card menu, so you must disable PC/SC while Jade is running as follows:

```
WNT Control Panel | Services | Smart Card Resource Manager stop
Control Panel | Services | CHIPDRIVE SCARD Service stop
W/98 Run msinfo32.exe, and choose System configuration utility from the tools menu.
In the Startup tab disable SCardSrv and TwkCardSvr then reboot.
```

Select Tools | Generate CAP File. Browse for your JAR file, click Parse, then enter the AID for the applet and package. The first five bytes of the package AID must be the same as the applet AID. The example applet uses "algsapp" and "algsa" for the applet and package, respectively. (" means use the ascii characters; you could also enter 61 6C 67 73 61 70 70, if you prefer.) Browse for your CAP file destination, entering the filename if it's the first time you're generating the CAP file. Then click on Generate CAP.

If the card already has an old version of the applet, select Card | Delete and remove the applet and then the package.

Then select Card | Load and Install, and browse for the CAP file. Enter your heap size and wait a bit. (The example applet has a heap of 1500.)

Summary

The main purpose of this article was to demonstrate how to use a Java Card smart card with Java. This was accomplished by giving a sample solution using the Java Cards built-in RSA cryptography functionality to generate an RSA signature. Obviously, you can do much more with a Java Card, but hopefully you will have a starting point of information and sources to get you started.

Future applications that integrate Java, smart cards, and cryptography are up to you! ☛

AUTHOR BIO

Andrew Webb is a staff engineer at Motorola World Wide Smart Card Division and holds a BS in applied mathematics from Carnegie-Mellon University.

andrew.webb@motorola.com

What's Online This Month...



tion servers, books, code protection, consulting services, database tools, education and training, testing tools, and more.

Salary Survey

Participate in our annual job/salary survey! We'd like to know the number of Java employees in your organization, the city and state you work in, what your job function is, how many years you've been in your field, and more.

The information you provide doesn't require any name or personal information and thus remains 100% anonymous. Results will be posted on our Web site and published in a future issue of **JDJ**.



JavaDevelopersJournal.com
www.javadevelopersjournal.com is your source for industry events and happenings. Check in every day for up-to-the-minute news and developments, and be the first to know what's going on in the industry.

Participate in our daily Live Poll and let your opinion be heard.

Java Buyer's Guide

Click here to go to the most-read Java resource on the Internet. Browse our listings for information on applica-



Java Jobs

Java Developer's Journal is proud to offer our employment portal for IT professionals. Through this site you have direct access to the best companies in the nation. If you're an information technology professional and are curious about the job market, demand privacy, and don't want to waste time, you've found the right site!

Simply type in the keyword, job title, and location and get instant results. You can search by salary, company, or industry.

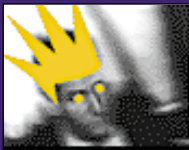
Need more help? Our experts can assist you with retirement planning, putting together a resume, immigration issues, and more. ☺



JDJ Readers' Choice Awards

Vote for your favorite Java software, books, and services in our annual **JDJ** Readers' Choice Awards, January 10 through May 30, 2001. Winners will be announced at JavaOne 2001 and presented at the International Conference for Java Technology - Fall Conference.





devicetop.com



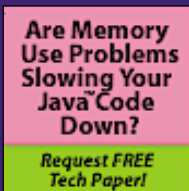
Sitraka Software



Get SonicMQ™
PROGRESS SOFTWARE



go Wireless
with air2web

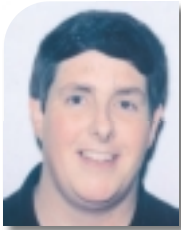


Are Memory Use Problems Slowing Your Java Code Down?
Request FREE Tech Paper!

EspressChart

by Quadbase Systems, Inc.

REVIEWED BY DON WALKER



AUTHOR BIO

Don Walker is cofounder and chief architect of Entice Software Corporation. He has over 15 years of experience in software development, working previously as a project lead at Simon & Schuster's Learning Technology Group.

don@enticesoftware.com



Quadbase Systems, Inc.
2855 Kifer Road
Suite 203
Santa Clara, CA 95051
Web: www.quadbase.com
Phone: 408 982-0835
Fax: 408 982-0838

Test Environment:
OS: Windows 95
Processor: 233MHz Pentium II
Memory: 32MB

Using charts makes complex data easier to comprehend. Unfortunately the decision to add charts to a Web site or an application doesn't necessarily make life easier for the designers and programmers responsible for displaying them. Depending on the complexity of the data, developers may encounter a seemingly endless series of questions. What type of chart is needed? Should the data be displayed horizontally or vertically? 2D or 3D? What increments should be used along the axis lines? What happens if the data changes?

EspressChart from Quadbase Systems, Inc., is a set of tools to help developers design and implement a variety of chart types. The primary tools that make up EspressChart include Chart Server, Chart Designer, Chart Viewer, and Chart API.

Chart Server provides user authentication as well as local and remote file access. Although Chart Server generally runs on a Web server, it's possible to run it on a stand-alone machine.

Chart Designer is a 100% Pure Java, interactive, front-end application that guides developers through the process of creating charts. Designers choose from a list of common chart types, then add customization - including rotating the chart to view it from different angles. Once a chart has been created, it can be exported in a variety of formats. In addition to common formats, such as .gif and .bmp, two formats specific to EspressChart are available. Charts exported as .cht (Chart Designer format) files are accessible to the Chart Viewer applet, as are charts exported as .tpl (the template format for Chart Designer). In addition, .tpl files save chart attributes and the data source, but not the actual data. Whenever a .tpl file is loaded, data for the chart is automatically updated.

Chart Viewer is an applet that allows users to view and manipulate charts remotely.

Finally, Chart API is a set of 100% Pure Java library functions that can work in conjunction with Chart Server or in a stand-alone mode to create and manipulate charts from within applets and applications. Chart API takes advantage of high-performance algorithms for displaying 3D objects. In fact, it can reproduce all the functionality of Chart Designer.

Creating a Chart

Suppose I want to produce a chart tracking the salary of Alex Rodriguez, who recently signed a 10-year, \$252 million contract to play shortstop for the Texas Rangers baseball team.

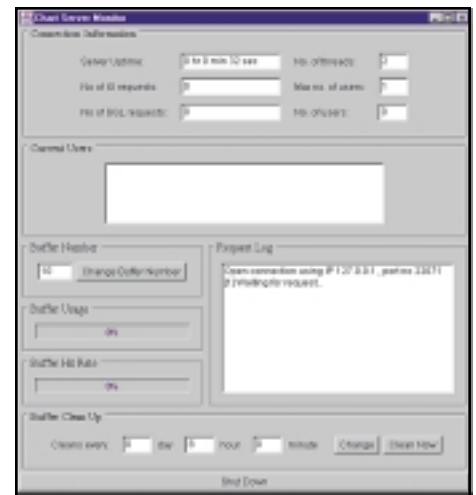


FIGURE 1 Chart Server monitor

Because Chart Designer requires Chart Server to be running, the first step is to launch Chart Server. I can accomplish this on my Windows 95 machine by simply executing server.bat. Several command line options are available, including one to turn the Chart Server monitor on or off. The Chart Server monitor displays information about the current status of the server, such as the number of users currently logged in (see Figure 1). In addition, it allows the administrator to change various settings and fine tune the performance of the server. For example, new to version 3 of EspressChart is a data buffering capability in which the administrator can control whether data requested from a database is stored in a buffer for faster access later or read from the data source each time.

Once Chart Server is up and running, Chart Designer can be launched as an applet by typing the proper URL into a browser or as an application by executing designer.bat. After the user has successfully logged on, five steps are required to create a new chart.

1. Specify a data source (database or data file).
2. Select a chart type.
3. Select suitable chart options.
4. Modify the chart design interactively.
5. Export the chart in the desired format.

If the data for a chart resides in a database, the Chart Wizard prompts you for the URL of the database, the database driver, user name and password, and a valid SQL statement to retrieve the data. In the example below, the data exists in a single data file called "Alex.txt", and I simply provide the name of the file when prompted by the Chart Wizard.

```
String, boolean, decimal, decimal  
Year, option, salary, deferred  
"2001", false, 21, 5  
"2002", false, 21, 4  
"2003", false, 21, 3  
"2004", false, 21, 3  
"2005", false, 25, 4  
"2006", false, 25, 4
```

```
"2007", false, 27, 4
"2008", true, 27, 3
"2009", true, 27, 3
"2010", true, 27, 3
```

In data files, the first row specifies the data types. The EspressoChart documentation lists 20 data-type keywords ranging from "boolean" and "int" to "date" and "timestamp". The second row contains the field names, and the remaining rows contain the records.

Once the data is ready it's time to decide precisely how it should be displayed. The Chart Wizard offers a choice of 17 2D and 13 3D chart types (see Figure 2). If the data is incomplete or incompatible with the chosen chart type, a dialog box appears with an explanation of the problem. The available chart types include:

- Column
- XY(Z) scatter
- Stack column
- Pie
- Stack area
- High low close open (HLCO)
- Surface chart (3D only)
- Overlay chart (2D only)
- Radar chart (2D only)

With the data source identified and the chart type selected, the next step is to precisely map the data from the data source to the chart. For example, the data file describing Alex Rodriguez's contract includes the salary total

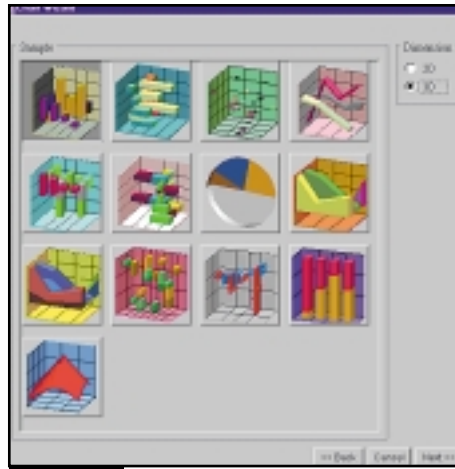


FIGURE 2 Choosing from standard chart types

for each year. Also included are whether the season is an option year and the amount of deferred money to be paid at a later date. This is valuable information, but we don't want to necessarily include it in this particular chart. Fortunately, through the Select Data Mapping dialog box, we can match a specific axis in our chart directly to a column of data in a data file or database.

Final Customization

At this point, designers concerned about the appearance of their charts can take advantage of the flexibility of Chart Designer to mod-

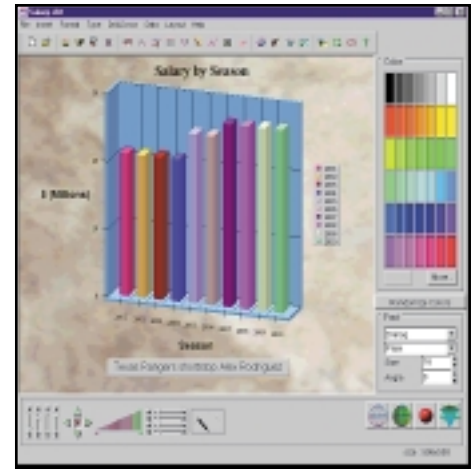


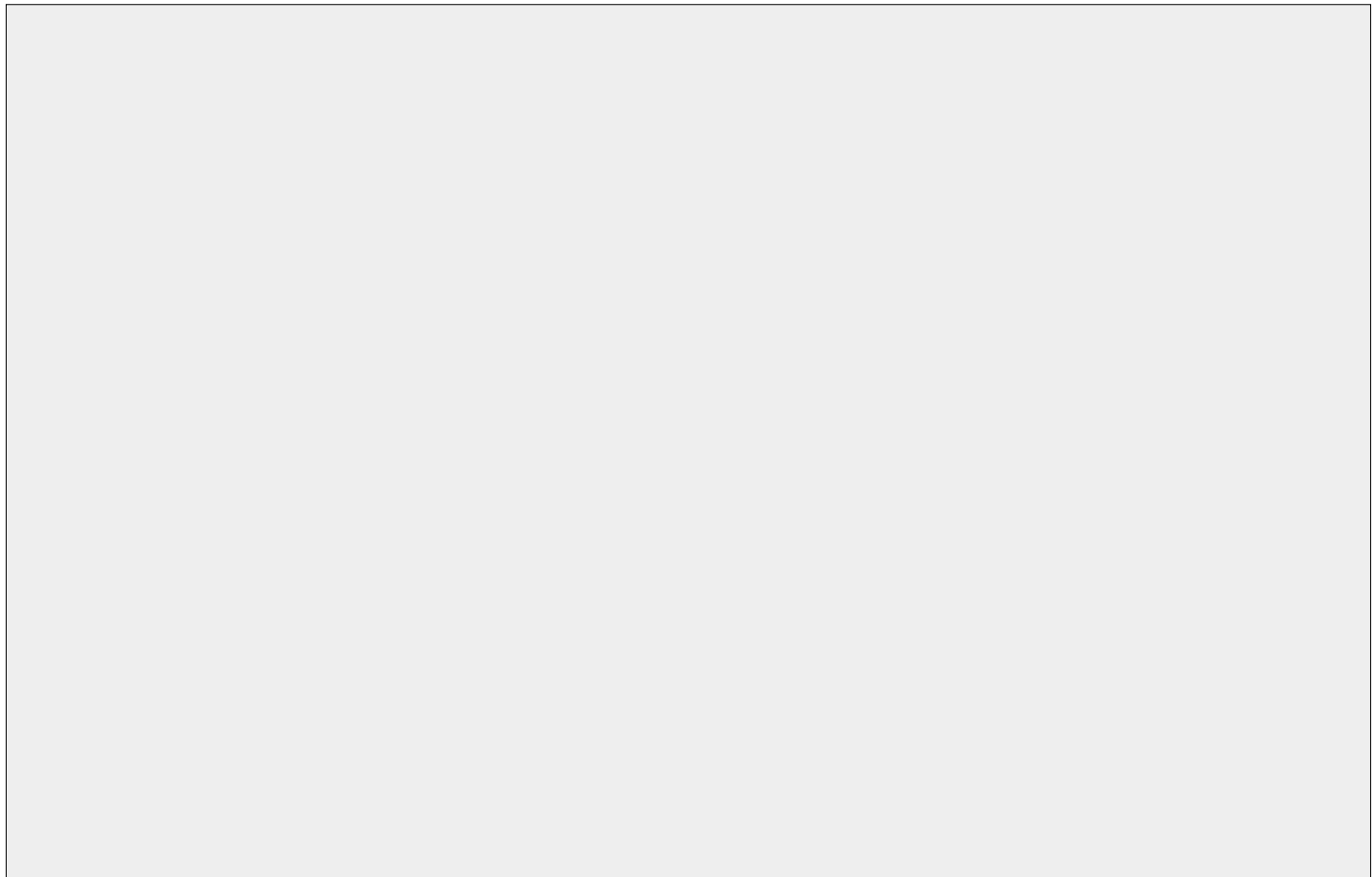
FIGURE 3 Chart Designer

ify the look of the chart, as well as add some functionality (see Figure 3).

A background image can be chosen for the chart and a title added. In fact, text can be included anywhere on the chart. Certain variables can also be included in the text for run time substitution. For example, these variables can represent the time, date, or name of a column of data mapped to a particular axis.

Customization isn't limited to adding items to the chart. Virtually any object can be removed. Titles, axis labels, even the axes themselves are not immune.

Further customization is possible for the items that remain. Color and font can be



changed. Objects can be moved. The limits of the axes can be changed. Even the thickness of the axis lines can be modified.

Finally, the entire chart can be resized and, in the case of 3D charts, rotated using the navigation panel at the bottom of the screen.

Not all changes made within Chart Designer are cosmetic. Significant functionality can also be added. Hyperlinks can be set up to allow the user to click on a data point and immediately branch to another chart or HTML page. For complex charts, the Drill-Down Wizard helps designers manage links between large numbers of charts by creating drill-down charts at runtime. Charts that contain frequently changing data can be configured to regularly check the database and display any changes.

When the designer decides the chart is complete, several nice features are available. When saving a chart, checkboxes exist for creating HTML and XML files. The HTML file contains code to display the saved chart, and the XML file contains the properties of the chart based on the `EspressChartAttributes.dtd` file. It's also possible to export the data of the chart to an XML file based on the `EspressChartData.dtd` file. This is particularly useful for charts based on data from multiple sources, since it allows the data to be stored in a single location.

Chart API

It's no exaggeration that Chart API can produce a chart with a single line of code. In this case, however, I needed two lines to display the salary chart as part of an applet. The call to the class method "setChartServerUsed" must be made prior to creating the chart object if, as in this case, the server isn't being used.

```
import quadbase.ChartAPI.*;
import java.awt.*;
import java.applet.*;
```

```
public class Alex extends Applet {
    public void init() {

        QbChart.setChartServerUsed( false );

        setLayout( new BorderLayout() );
        add( "Center",new QbChart(this,"Salary.cht") );
    }
}
```

Because the salary chart is in 3D, the navigation panel is included in the applet, allowing users to rotate the chart to view it from various angles within the applet without the need for me to write any additional code.

Since Chart API can re-create all the functionality of Chart Designer, I can use it to further modify the chart by changing the title, for example. I can also use it to export the chart as a static image. In this way developers can use JavaServer Pages or servlets to generate charts on the server side before displaying a Web page on the static image and the corresponding map file (if any) dynamically.

Summary

The Chart Designer, despite its power and flexibility, is remarkably intuitive. When I decided I didn't like the background image, I double-clicked on it and was immediately presented with a list of potential replacements. Moving objects, such as the legend, simply involves clicking and dragging.

Still, documentation is important, and the `EspressChart` documentation is outstanding. The chapter on Chart API has generous portions of sample code, and the chapter on Chart Designer contains detailed descriptions of all of the standard chart types, including guidance on when to use a particular type of chart.

More information, plus the opportunity to download an evaluation copy of `EspressChart`, is available at www.quadbase.com.

WebLogic Collaborate 1.0



BEA

WebLogic Collaborate is a layered product that builds on BEA's WebLogic application server product. While it's possible to build your own B2B solution from the ground up, Collaborate reduces your time-to-market by offering a set of prebuilt components. It leverages existing Web technology standards as well as burgeoning commerce standards.



Many B2B sites are built around the concept of an exchange. Collaborate implements the exchange concept through something they call Collaboration spaces, or "C-Space". Because each C-Space can support different trading protocols, a single C-Hub can handle both RosettaNet and EDI.

Prepackaged B2B solutions are likely to offer more out-of-the-box functionality than Collaborate. However, Collaborate's advantage may lie in its integration with the Application Server.

Contact: www.bea.com

Internet File System



Oracle

Oracle has released an updated version of their Internet File System product with Oracle8i Release 3.0 and the Oracle 9iAS application server. IFS replaces the native file system on your server with the advanced Oracle



database. End-user clients work with familiar interfaces to see and edit database-held documents and media as files

and folders. The system supports a variety of common access protocols that give clients the ability to store, manage, and search documents, presentations, multimedia, Web pages, and XML files across a variety of devices. IFS provides a secure layer in which to store all of your data, both structured and unstructured. However, IFS is more than just a replacement for your file system - it's actually a complete content management system in its own right.

One of the keys to IFS is its extensibility. This latest release of IFS includes an improved Java-based API, and this is one of its strengths.

Contact: www.oracle.com

Web BusinessManager Suite 6.0



SpaceWorks

BusinessManager Suite 6.0 is designed to help companies automate all their B2B activities over the Web. It incorporates the J2EE architecture and offers support for EJBs, JSPs, Servlets, JNDI, JMS, RMI, and JTS. Business-



Manager is offered as a comprehensive

prebuilt product set - targeting those organizations that want to buy an off-the-shelf solution.

The suite spot for the product is its support for complex order management that can be implemented on a module-by-module basis: order management, marketing programs, customer service, billing, reporting, and B2B integration. You can customize the product using J2EE technologies, but most of the customization will be at the business-processing level. Although BusinessManager is written with J2EE technology, it's not an application development environment. You'll have to spend some time integrating BusinessManager into your back-office applications. It's clearly a solution for the high end of the market.

Contact: www.spaceworks.com

Metrowerks and Partners to Deliver Wireless Solution

(Santa Clara, CA) – Metrowerks is teaming up with technology partners to provide an out-of-the-box solution to help developers create enterprise-ready applications for wireless devices. The solution, which Metrowerks plans to make available in the first quarter of 2001, will combine the familiar CodeWarrior IDE and toolset with PointBase Inc.'s database, Eliad Technologies' iSmartGrid, and NewMonics' PERC virtual machine.



www.pointbase.com
www.eliad.com
www.newmonics.com
www.metrowerks.com

Silverstream Software Achieves Full J2EE Certification

(Billerica, MA) – SilverStream Software, Inc., has successfully completed Sun Microsystems' J2EE certification test suite assuring cross-platform compatibility between J2EE specification-certified vendors. To reach compatibility, SilverStream passed more than 5,000 rigorous compliance tests.



www.silverstream.com

Integration Enhances Open Text's Collaborative Solutions

(Waterloo, ON) – Open Text Corporation and Cimmetry Systems announced that AutoVue and Panoramic! for Java now integrate with Livelink, providing Livelink customers with enhanced tools for their knowledge management and enterprise resource planning solutions.

This integration enables users to securely access drawings and documents from any Web browser, add graphical or textual information to documents and drawings independent of the authoring application without altering the original, and create a work-



flow for making revisions to engineering drawings and business documents.

www.cimmetry.com
www.opentext.com



O'Reilly Releases Java Message Service

(Sebastopol, CA) – Any developer or system architect who has a need to connect applications will benefit from the information in *Java Message Service* by



David Chappell and Richard Monson-Haefel. This book demonstrates how to build applications using the

point-to-point and publish-and-subscribe models, how to use features such as transactions and durable subscriptions, and how to use messaging within EJBs. It also introduces a new EJB type, the MessageDriven-Bean, that's part of EJB 2.0, and discusses integration of messaging into J2EE.

www.oreilly.com/catalog/javmesser/

O'REILLY Online Catalog

Insignia Solutions Teams with ProSyst USA

(Fremont, CA) – Insignia Solutions and ProSyst USA have partnered to provide ProSyst's customers with Insignia's accelerated Java-

compatible technology. Insignia's Jeode platform



will integrate with ProSyst's mBedded Server software to give

developers the technology they need to deploy embedded Java applications that are fast, small, and functional.

www.prosyst.com
www.insignia.com



NexPrise Selects Cimmetry for Visualization Capabilities

(Santa Clara, CA) – Cimmetry Systems Inc.'s online visualization tool, AutoVue for Java, is an additional offering to the latest release of NexPrise ipTeam, version 4.0. The technology combination improves and eases communications between virtual team members connected across organizational and geographical boundaries.

www.cimmetry.com
www.nexprise.com



IBM developerWorks Teams with Flashline.com

(Cleveland, OH) – Flashline.com Inc.'s products and services for component-based development have become available through



IBM developerWorks, a Web site that provides developers with free content on open standards-based development and cross-platform technologies. The cobranded site, Components Marketplace, is the result of an agreement made last June between Flashline and IBM developerWorks.

www.ibm.com/developerWorks



SYS-CON Names Agnes Vanek Corporate VP of Circulation

(Montvale, NJ) – Veteran magazine industry executive Agnes Vanek has joined SYS-CON Media, Inc., as corporate vice president of circulation.

"We are delighted to have Agnes Vanek on board," said Fuat Kircaali, founder and CEO of SYS-CON Media. "Agnes will have a critical role in the launch of our most recent and largest title, *Wireless Business & Technology*. She will redefine an aggressive circulation strategy for our existing titles including *Java Developer's Journal* and *XML-Journal* and help us successfully launch a number of new world-class titles for our technology readers around the globe."

"I am very excited to join the SYS-CON team and look forward to being part of the group that has created the most successful technology magazines in recent years," said Vanek. "*Wireless Busi-*

ness & Technology aims to be the world's leading publication serving the wireless industry and Internet tech-

nology developers utilizing new and emerging wireless technologies."

Vanek started her high-tech career in 1983 at Ziff-Davis as senior circulation manager for *PC Week* and *MacWeek*. Her 18-year professional accomplishments include managing *CommunicationsWeek* and *CommunicationsWeek International* at CMP. During Vanek's

eight-year tenure at IDG, her circulation strategy played a key role in making *Federal Computing Week* the lead publication in the government IT market. In 1996 Vanek successfully launched FCW's sister publication, *civic.com*.



Sitraka Establishes Regional Office in San Francisco

(Toronto, ON) – Sitraka Software (formerly KL Group) has launched its West Coast office in San Francisco, California. The initiative serves as the first step



in growing Sitraka's Silicon Valley presence and in enabling it to better serve the needs of its global customers headquartered in California.

The California office will be headed by Michael Murray, western regional manager, who will be responsible for sales management and new business development activities.

www.sitraka.com

Sheridan, ProtoView Merge to Form Infragistics

(Cranbury, NJ / Melville, NY) – Sheridan Software Systems, Inc.,

and ProtoView Development Corporation have announced their merger. The new company, Infragistics, Inc., will be headquartered in Cranbury, New Jersey.

Infragistics has merged COM products from both the Sheridan and ProtoView product lines to create the Infragistics UltraSuite.



Infragistics will continue to support the other COM products from the two former companies, and publish and move forward with InterAct, PowerChart, the JSuite, and the JFC Suite, all formerly published under the ProtoView label.

www.infragistics.com

Compoze Releases Harmony Component Suite 1.1

(Philadelphia, PA) – Compoze Software, Inc., announces Harmony Component Suite 1.1, a collection of EJB components



offering support for multiple application servers, including

Allaire JRun, BEA WebLogic, the J2EE Reference Implementation, and the Orion Application Server. Database support includes Oracle, Microsoft SQL Server, Sybase, and Cloudscape.

www.compoze.com

B2B ITS Corp. Releases .FIXantenna Engine As Open Source

(Stamford, CT) – B2B ITS Corp. has released FIXantenna Engine – the first open source FIX Engine offered under an Apache-style license.

FIXantenna Engine's release coincides with the launch of FIXantenna, a suite of FIX protocol-related products for the financial services industry. In addition to the FIX Engine, FIXantenna includes additional external CORBA services and a FIX Message Browser. FIXantenna Engine can be

downloaded at <http://fix.btobits.com>.



To B2B or Not To B2B — and Other Questions

Flexibility is the key

WRITTEN BY
BILL BALOGLU &
BILLY PALMIERI



This dramatically changing, high-tech landscape is causing fear and confusion in Java engineers. What sensible, intelligent person wouldn't be shaken by the current state of affairs in the dot-com world?

As a Java engineer, what are your options? What does it all mean? And what should your next step be?

AUTHOR BIOS

Bill Baloglu is a principal at Object Focus (www.ObjectFocus.com), a Java staffing firm in Silicon Valley. Prior to ObjectFocus, Bill Baloglu was a software engineer for 16 years. He has extensive OO experience and has held software development and senior technical management positions at several Silicon Valley firms.

Billy Palmieri is a seasoned staffing industry executive and a principal of ObjectFocus. Prior to ObjectFocus, he was at Renaissance Worldwide, a multimillion dollar, global IT consulting firm, where he held several senior management positions in the firm's Silicon Valley operations.

For example, on the consulting front over the past few years, many skilled engineers took a cut in their usual contracting pay to join a dot-com in exchange for huge stock options. They took the risk in exchange for a shot at huge rewards – and the chance to be in on the ground floor of the next eBay.

Now many of those start-ups have crashed and burned, leaving engineers with thousands of worthless stock options – or no job at all. From their perspective, they were taken for a ride.

What's the solution now? Some engineers who were burned in the dot-com crash feel it would be best to join a huge corporation, such as Sun, HP, or Cisco, for the sense of security working for a large company can provide.

Maybe I won't make as much money as I did contracting or get as many stock options as I did at that dot-com; at least I'll have an inner feeling of security knowing the company won't go belly-up at a moment's notice.

But ultimately no position in high tech is totally secure. The days of getting a job at General Motors and knowing that you'd work there for 30 years are long gone.

Dot-coms are crashing and burning all around us. Yesterday's "hot" companies are seeing their stock values plummet, triggering massive layoffs. Even "sure bet" companies seem to be going out of business everywhere you look.

Even at large companies major projects get scrapped, groups get reorganized or eliminated, and the door to job security keeps revolving.

Those familiar with the Web generation of technology know that technology changes every 18 months, ways of doing business change, and your skill sets can quickly become obsolete.

Before totally discounting opportunities at smaller, entrepreneurial companies, look closer at what can be learned from the current dot-com meltdown:

- The Internet isn't going away.
- Companies that support and do business on the Internet are here to stay.
- B2Bs and infrastructure companies are faring better than B2C companies, which became overvalued based on market share versus potential for real revenue.
- Venture capitalists that once invested like mad in brand-new start-ups are now focusing on third rounds of funding in companies that are more secure deals.
- Venture capital funds are at their highest levels, and VCs still have plenty of money to invest. However, we're starting to see larger amounts of VC investments going into a smaller number of more select companies.
- Pre-IPOs in their post-second round of funding with solid business models and management teams are still a good bet.

The illusion that every high-tech company is failing is just that: an illu-

sion. The percentage of new companies that have failed is the same as it is in any given year. The difference is in the huge amount of new companies that went into business, which resulted in a higher amount of total companies that failed.

But what's the bottom-line result of all this for Java engineers? Rates and salaries are likely to go down from where they were in the crazy dot-com "boom days," but should stabilize as the market gets healthier.

- Senior engineers will always be in demand and aren't likely to see significant changes.
- Intermediate engineers may see some changes in their rates, and it may take about two weeks for them to find a job (as opposed to five days).
- Junior engineers are most likely to be laid off first and may see pay rates go down (after all, there are a lot more junior engineers). This is an ideal time to enhance your skill sets with recent technologies that will increase your marketability.

Across the board, sensible Java engineers will consider being flexible about their current rates or salaries in exchange for long-term opportunities in a market that's headed for increasing stability. ☘

billb@objectfocus.com

billp@objectfocus.com



Call Ron Perretti Today! 201-802-3028

WRITTEN BY AJIT SAGAR



IMHO: Blueprinting Java

Last year Sun came out with a new set of design guidelines for building enterprise applications using enterprise Java APIs. These APIs are available as a set of documents called the *J2EE Blueprints*. They include architectural design guidelines for developing enterprise applications using the Java 2, Enterprise Edition APIs.

The Silver Bullet

The primary benefit of the Blueprints is that after five years and several releases of Java platform products and APIs, there's finally a comprehensive story of how all these technologies offered by Java can plug and play together in enterprise-level applications. Using the Blueprints as guidelines also helps architects and developers make choices between alternative technologies and products, based on the constraints of their business and operating environments.

The J2EE Blueprints address enterprise application development using the design pattern MVC (Model-View-Controller) to build the underlying framework. *Designing Enterprise Applications with the Java 2 Platform, Enterprise Edition* (Addison-Wesley) covers the Blueprints in detail. A PDF of the Blueprints as well as a sample "Pet Store" application can be downloaded from Sun's Web site, <http://java.sun.com/j2ee/download.html>. The sample illustrates how the Blueprints can be applied in a distributed business application.

The Whole Enchilada?

This is great stuff. You now have a single source to get all the information you need to develop enterprise-level, distributed, transactional applications using only Java technologies. Select your application server, download the appropriate APIs, and off you go. If you can, use Java APIs to create all the building blocks in your application. If you can't, there are well-defined integration points to connect to the outside world.

The real questions you should ask yourself are how much of this do you want to build in-house, and how much do you want to buy off the shelf? If you were developing the Pet Store application in the real world, you could

probably build everything using your development resources. However, if you're dealing with applications that span multiple business scenarios, chances are you'll soon run into issues such as resource allocation and maintenance if you decide to build all the pieces yourself. Of course if you're in the business of building frameworks and application servers, it's a great idea to do it all yourself. However, that will be your main product, not the business applications you build on top of such frameworks.

Let's look at the presentation layer. You can use a combination of JSP, servlets, and XML to create your presentation layer. JSP can be used to create the presentation templates and guide the flow of pages by designing the layout manager for your site. Servlets can send the appropriate content into syndicated columns. And XML serves as a great format for exchanging data with the outside world. The combination of these technologies can be used to create a presentation layer for your application.

What happens when you want this framework to be generic so it can be applied across a variety of business scenarios? How much effort will be required to further abstract your design to provide templates that can be configured for different applications? This is where you'll start running into maintenance and resource problems.

If It Sounds Too Good To Be True . . .

Don't get me wrong. It's not that the Blueprints mislead the development community into believing that everything should be done in-house. It's just that they can be interpreted in different ways. Typically, if you were working on applications that span several business scenarios and applications, you would depend on technology vendors such as application server providers to implement the frameworks that make it all possible. And dare I say it, you would also look outside the Java world for some of your needs. A large part of the existing presentation and personalization products in the market are built on Web scripting technologies that complement Java environments. For example, companies such as Allaire, BroadVision, Vignette, and ATG provide the frameworks required to build such applications. That's the very reason they're in business. ☘

ajit@sys-con.com

AUTHOR BIO

Ajit Sagar is the founding editor and editor-in-chief of XML-Journal. A senior solutions architect with VerticalNet Solutions based in San Francisco, he's well versed in Java, Web, and XML technologies.