



JAVA™ DEVELOPER'S JOURNAL

The World's Leading Java Resource

August 2002 Volume:7 Issue:8

JAVADEVELOPERSJOURNAL.COM

FULL CONFERENCE PROGRAM

THE LARGEST WEB SERVICES CONFERENCE & EXPO IN THE WORLD

WEB SERVICES EDGE

INTERNATIONAL WEB SERVICES CONFERENCE & EXPO

INTEGRATING APPLICATIONS
CONNECTING ENTERPRISES

JDJEDGE CONFERENCE: October 1-3, 2002

WMLEDGE CONFERENCE: October 2-3, 2002

CALL TODAY!
201-900-3000

SAVE \$200

INSIDE PAGE 77

From the Editor
Alan Williamson pg. 7

J2EE Editorial
Ajit Sagar pg. 9

J2EE FAQ
pg. 10

J2SE Editorial
Jason Bell pg. 36

J2ME Editorial
Jason R. Briggs pg. 54

Cubist Threads
Blair Wyman pg. 102

RETAILERS PLEASE DISPLAY
UNTIL OCTOBER 31, 2002

\$5.99US \$6.99CAN



SYS-CON MEDIA



Multichannel Communication: Using Notification Servers in J2EE Applications *Sunil Makhijani*

Free up your development time **12**

Feature: Making Your EJBs Polymorphic *John Musser & Bernhard Lenz*
Combine inheritance with EJBs **20**

Standards: J2EE Frameworks - Extend and Submit *Ted Farrell*
Maximizing productivity across your organization **28**

Reusable Components: A List Tool for All Applications *Teresa Lau*
Designing for reuse and code maintainability **38**

Platform Support: Does J2ME Have Its Legs Yet? *Matthew Ferris*
How to take advantage of mobile device programming **56**

Feature: Wireless J2ME Applications with Java and Bluetooth *Bruce Hopkins*
The essential components **60**

JDJ Labs: Small Worlds 1.5 *Dale Churchett*
by Information Laboratory, Inc. **68**

Show Report: Web Services Edge 2002 East Conference & Expo *Steven Berkowitz*
Show wrap-up **70**

sonic
www.sonic.com

zerog
www.zerog.com

ap

WWW.

ple
apple.com

bea
www.bea.com

INTERNATIONAL ADVISORY BOARD

- CALVIN ALSTIN (Lead Software Engineer, J2SE Linux Project, Sun Microsystems)
- JAMES DUNCAN DAVIDSON (JavaServlet API/MIP API, Sun Microsystems)
- JASON HUNTER (Senior Technologist, CollabNet) • JON S. STEVENS (Apache Software Foundation) • RICK ROSS (President, JavaLobby) • BILL ROTH (Group Product Manager, Sun Microsystems) • BILL WILLETT (CEO, Programmer's Paradise)
- BLAIR WYMAN (Chief Software Architect IBM Rochester)

EDITORIAL

- EDITOR-IN-CHIEF: ALAN WILLIAMSON
- EDITORIAL DIRECTOR: JEREMY GEELAN
- EXECUTIVE EDITOR: NANCY VALENTE
- J2EE EDITOR: AJIT SAGAR
- J2ME EDITOR: JASON R. BRIGGS
- J2SE EDITOR: JASON BELL
- PRODUCT REVIEW EDITOR: JIM MILBERY
- FOUNDING EDITOR: SEAN RHODY

PRODUCTION

- VICE PRESIDENT, PRODUCTION AND DESIGN: JIM MORGAN
- ASSOCIATE ART DIRECTOR: LOUIS F. CUFFARI
- EDITOR: M'LOU PINKHAM
- MANAGING EDITOR: CHERYL VAN SISE
- ASSOCIATE EDITORS: JAMIE MATUSOV
GAIL SCHULTZ
JEAN CASSIDY
- ASSISTANT EDITOR: JENNIFER STILLEY
- ONLINE EDITOR: LIN GOETZ
- TECHNICAL EDITOR: BAHADIR KARUV, PH.D.

WRITERS IN THIS ISSUE

- BILL BALOGLU, JASON BELL, STEVEN BERKOWITZ, JASON BRIGGS, DALE CHURCHETT, TED FARRELL, MATTHEW FERRIS, BRUCE HOPKINS, DOV KRUGER, THERESA LAU, BERNHARD LENZ, SUNIL MAKHIANI, JOHN MUSSER, BILLY PALMIERI, NORMAN RICHARDS, AJIT SAGAR, ALAN WILLIAMSON, BLAIR WYMAN

SUBSCRIPTIONS:

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

SUBSCRIPTION HOTLINE: SUBSCRIBE@SYS-CON.COM

COVER PRICE: \$5.99/ISSUE

DOMESTIC: \$49.99/YR. (12 ISSUES)

CANADA/MEXICO: \$79.99/YR. OVERSEAS: \$99.99/YR.

(U.S. BANKS OR MONEY ORDERS). BACK ISSUES: \$10/EA., INTERNATIONAL \$15/EA.

EDITORIAL OFFICES:

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

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

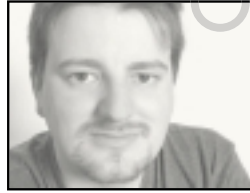
JAVA DEVELOPER'S JOURNAL (ISSN#1087-6944) is published monthly (12 times a year) for \$49.99 by SYS-CON Publications, Inc., 135 Chestnut Ridge Road, Montvale, NJ 07645. Periodicals postage rates are paid at Montvale, NJ 07645 and additional mailing offices. POSTMASTER: Send address changes to: JAVA DEVELOPER'S JOURNAL, SYS-CON Publications, Inc., 135 Chestnut Ridge Road, Montvale, NJ 07645.

© COPYRIGHT:

Copyright © 2002 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 Carrie Gebert, carrieg@sys-con.com. SYS-CON Publications, Inc., reserves the right to revise, republish and authorize its readers to use the articles submitted for publication.

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.



ALAN WILLIAMSON EDITOR-IN-CHIEF

Only Now Is Java Coming Fully Alive

This past June, the crew and I spent a week in one of the best cities in the world. New York City played host to **SYS-CON's Web Services Edge Conference & Expo**, where all the major players in the Web services market come under one roof to talk and debate the emergence of this new wave of technology. It was good to catch up with people and I thoroughly enjoyed grabbing some quality face-to-face time with a number of authors, including Joey Gibson and Rick Hightower, to name but a few. Their insight into the current trends and technologies was most enlightening, and, hopefully, I'll be able to snaffle a guest editorial from them at some point.

This is the real reason to go to these types of shows: the people, listening to what they have to say and learning what projects are currently on the go. It seems that opinion on the whole Web services revolution is still divided. I had the dual pleasure of chairing the "Supercharging Web Services with Java" panel and sitting in on the "J2EE vs .NET" panel. For example, when Simon Phipps (Sun's chief technology evangelist) was asked to comment on the Web services movement for the last 12 months, the first thing he did was clarify his definition of Web services. Personally, I think this was more to clarify Sun's position and differentiate themselves from Microsoft's view of Web services.

My most startling observation was that .NET managed to creep into every conversation, with everyone offering their take on the whole good versus evil war. Rick Ross, never one to shy away from controversy, stirred the pot a bit by claiming that Microsoft had the greatest virus ever built: the Windows Update. It's used to ship service packs and patches to desktops; competing with that is going to prove difficult. He's right. Have you updated your Windows recently? Notice the .NET Framework is now listed as one of the options you can install. This will definitely

move the penetration of .NET somewhat.

Let's be completely honest here: Microsoft isn't going away. No matter what we say regarding their skullduggery tactics, .NET is here and we'll have to learn to work with it, not against it. Our greatest downfall as a Java community would be to ignore it. If we're going to win this game, we have to weigh up our opponent properly, and take them on full tilt, as opposed to standing on the sidelines shouting abuse.

To a large degree we've lost the battle on the desktop, or at least the Microsoft Windows desktop. In fact, it's arguable whether we were ever in the running. Swing has done us no favors whatsoever, and while the work Apple has done to make Swing run like lightning is fantastic, it's a little late coming to the party. We needed that performance a number of years ago. This was a hotly debated subject on the panels, with even the likes of Ross and Phipps conceding that this race has been largely lost.

However, before you go shelving your Java books, it's important to remember where the majority of the Java development is happening and why. The server side is where Java has a very strong foothold, and one that I don't see being displaced no matter how many marketing dollars Microsoft spends with their 1° of separation campaign. Just not going to happen. Java has proved itself time and time again as the only scalable, portable, cost-effective solution available. Take this power and also transport it to the mobile space, and you'll discover another revolution taking place that will have Java firmly placed to lord over this arena.

A lot of industry pundits are already writing Java's eulogy, proclaiming Microsoft the winner. It angers me to read such blasphemous articles from respected sources. Java is far from dead: on the contrary, only now is Java coming fully alive. In my view, you ain't seen nothing yet. ☼

AUTHOR BIO

Alan Williamson is editor-in-chief of Java Developer's Journal. During the day he holds the post of chief technical officer at n-ary (consulting) Ltd, one of the first companies in the UK to specialize in Java at the server side. Rumor has it he welcomes all suggestions and comments.

alan@sys-con.com

metroworks
metroworks.com

9

Talking About My Generation

Though technology brings to fruition concepts that were conceived of only a few decades ago, our expectations of technology far exceed the speed at which it makes solutions available.

by Ajit Sagar

10

J2EE FAQ

The answers to your J2EE questions

12

Using Notification Servers in J2EE Applications

With the emergence of new notification channels (such as WAP phones, instant messaging applications, and SMS pagers), sending notifications has become more complicated. Now applications have to support an ever-changing set of communication channels that end users would like to be notified about.

by Sunil Makhijani

20

Making Your EJBs Polymorphic

Inheritance and polymorphism are two of the most fundamental concepts in the object-oriented design world. They are used extensively in all Java applications, except J2EE apps using EJBs. The goal is to combine some advantages of inheritance such as code reuse and polymorphism with the advantages of EJBs, such as distributed transactional components.

by John Musser & Bernhard Lenz

28

J2EE Frameworks – Extend and Submit

As the J2EE platform continues to grow and gain ground in corporations and the battle between Java and Microsoft .NET intensifies, more and more companies are looking for help in building their J2EE applications.

by Ted Farrell



AJIT SAGAR J2EE EDITOR

Talking About My Generation

I'm sure you've heard many of the cannibal jokes. One of my favorites is a news flash in a cannibal tribe announcing the invention of the "pressure cooker": "We have news of a device that cooks a man within minutes, and even lets out a whistle when it's done." Though technology brings to fruition concepts that were conceived of only a few decades ago, our expectations of technology far exceed the speed at which it makes solutions available. Often when I'm in the middle of architecting an *n*-tier application or framework using J2EE technologies, I find myself asking questions, such as: "What if I didn't have to write a line of code to achieve this," or "can I just cut-and-paste this into my next solution?"

My colleagues are often amused by my faith in the fine art of "cut-and-paste." However, when things are down to the wire, the speed at which existing code can be slightly modified through pure editing expertise is sometimes unmatched. Of course, this can lead to redundant code, but that's one of the joys of three-month releases. When there's a deliverable next week, software engineering takes a backseat. That's the harsh reality of the software-development life cycle that many of us are subsumed in. The key is to go back and reengineer your solution. Of course, those days when you could just as easily engage external consultants to do so disappeared when the technology bubble burst.

Note that I am not advocating short-circuiting the software engineering cycle or circumventing the development process. Every time a shortcut is taken, it should be documented, and the appropriate design should be suggested – to be revisited at a later time. Fortunately, the latest Java IDEs offer several tools that alleviate the pain of this unconventional design cycle, such as refactoring support and reverse-engineering source code into modeling environments such as UML. The latest IDEs that I've worked with are IntelliJ's IDEA and

Borland's JBuilder. Both offer these features, as do other competing products. Add unit and regression testing and source code control to the mix, and the proposed design cycle is almost achievable.

However, as software architects and developers, part of our job is to create the next level of abstraction. A software architect has substantial technology expertise as well as business-domain expertise. A J2EE architect utilizes his or her experience in applying Java's distributed technologies to a solution in the primary application domain. This involves creating not only reusable components and code, but also automating the process of software development for the next cycle.

I'd like to cite the example of one of my recent projects. Our mission was to build an application for the wireless domain. The choice of J2EE technologies was an obvious one, given our environment and our expertise. With the advent of EJB 2.0, the Enterprise JavaBeans-based implementation fit the bill (as opposed to a mix of JSP, servlet, JDBC, and EJB). The approach we took in our design was to automate the process of creating the middle-tier components (the EJBs), the hardest tier to build, test, and deploy. This included both the entity as well as the "accessor" session beans. We added utilities to generate components that leveraged the app server tools with the right level of business-domain components, so that we could replicate, enhance, and modify these components without rewriting too much code. The utilities also generate the database schemas and the constraints on the database. Although this effort cost us a precious couple of months in our release cycle, over a six-month cycle we were able to build an application that can now be applied across a variety of customers in our business domain.

Although there are no silver bullets, designing for reuse and automating component generation can take you a long way toward achieving your ultimate objective. ☛

ajit@sys-con.com

AUTHOR BIO

Ajit Sagar is the J2EE editor of JDJ and the founding editor of XML-Journal. Ajit is the director of engineering with Controlling Factor, a leading B2B software solutions firm based in Dallas, and is well versed in Java, Web Services, and XML technologies.

WHAT TYPE OF JAVA OBJECTS DO DEVELOPERS CREATE IN EJB DEVELOPMENT?

For each Enterprise JavaBean in your application, you need to extend the appropriate home and remote interfaces (local/remote), and provide an implementation for the appropriate enterprise bean (entity/session/message-driven). The local interfaces and message-driven beans were introduced in EJB 2.0. The J2EE FAQ in a previous *JDJ* (Vol. 7, issue 5) covered the interfaces provided by the J2EE application server vendor. You may want to refer to that FAQ to set the context for this month's.

The figure below illustrates the classes EJB developers need to implement to build their applications.

The top box shows the interfaces provided by the J2EE application server. The bottom box shows the classes that need to be implemented by EJB developers. *Note:* You don't need to implement all the classes shown in the figure. Implement only what's required by the application. For example, you may not need a message-driven bean for your application, or you may choose to use session beans directly and access the database via JDBC rather than entity beans.

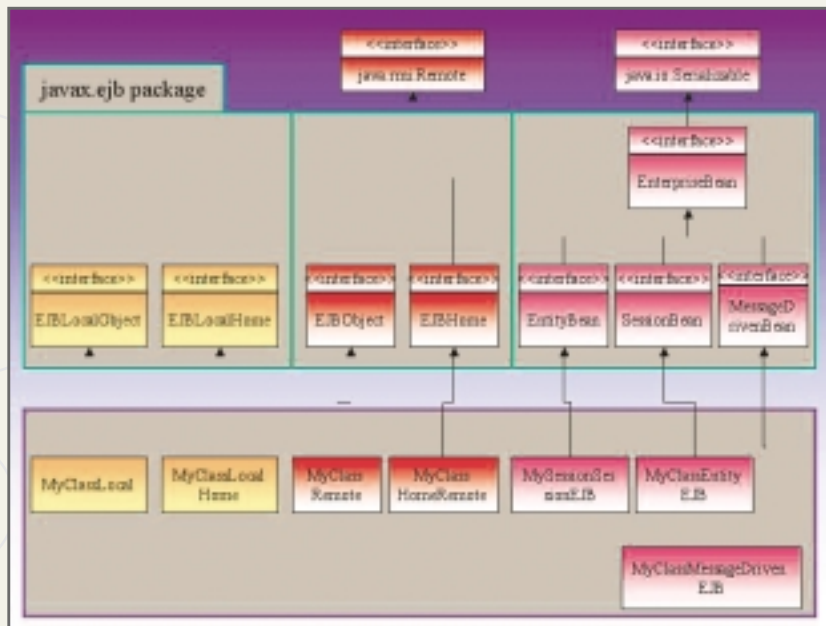
Typically, to write an EJB component, you need to extend two interfaces and implement one

class. However, if you're writing a message-driven bean, implement the EJB class only, not the interfaces. Since message-driven beans are the exception to the rule, we'll discuss them after we look at the classes/interfaces that developers need to code.

In EJB 2.0, before coding your EJB component (session or entity bean) you need to determine whether you want to make its methods available through a local or a remote interface, i.e., will your beans be colocated in the same application server or will they communicate across the network? In EJB 1.1, local interfaces didn't exist, so you always had to create the remote interface. As you can see in the figure, the two interfaces you need to implement are the home (EJBHomeLocal/EJBHome) and object interfaces (EJBObjectLocal/EJBObject) for the bean.

Let's assume you're creating a new EJB component class called MyClass. You would have to create the appropriate interfaces for the class to be used, then you'd have to implement the class itself.

For message-driven beans, only the EJB class needs to be coded. The EJB interfaces are not needed since there are no direct calls from the client. ☛



- PUBLISHER, PRESIDENT, AND CEO
FUAT A. KIRCAALI fuat@sys-con.com
COO/CEO
MARK HARABEDIAN mark@sys-con.com
VICE PRESIDENT, BUSINESS DEVELOPMENT
GRISHA DAVIDA grisha@sys-con.com
ADVERTISING
SENIOR VICE PRESIDENT, SALES AND MARKETING
CARMEN GONZALEZ carmen@sys-con.com
VICE PRESIDENT, SALES AND MARKETING
MILES SILVERMAN miles@sys-con.com
ADVERTISING SALES DIRECTOR
ROBYN FORMA robyn@sys-con.com
ADVERTISING ACCOUNT MANAGER
MEGAN RING megan@sys-con.com
ASSOCIATE SALES MANAGERS
CARRIE GEBERT carrieg@sys-con.com
KRISTIN KUHNLE kristen@sys-con.com
ALISA CATALANO alisa@sys-con.com
LEAH HITTMAN leah@sys-con.com
EDITORIAL
EXECUTIVE EDITOR
NANCY VALENTINE nancy@sys-con.com
EDITOR
M'LOU PINKHAM mpinkham@sys-con.com
MANAGING EDITOR
CHERYL VAN SISE cheryl@sys-con.com
ASSOCIATE EDITORS
JAMIE MATUSOV jmie@sys-con.com
GAIL SCHULTZ gail@sys-con.com
JEAN CASSIDY jean@sys-con.com
ASSISTANT EDITOR
JENNIFER STILLEY jennifer@sys-con.com
ONLINE EDITOR
LIN GOETZ lin@sys-con.com
PRODUCTION
VICE PRESIDENT, PRODUCTION AND DESIGN
JIM MORGAN jim@sys-con.com
LEAD DESIGNER
LOUIS F. CUFFARI louis@sys-con.com
ART DIRECTOR
ALEX BOTERO alex@sys-con.com
ASSOCIATE ART DIRECTORS
CATHRYN BURAK cathyb@sys-con.com
RICHARD SILVERBERG richards@sys-con.com
AARATHI VENKATARAMAN aarathi@sys-con.com
ASSISTANT ART DIRECTOR
TAMI BEATTY tami@sys-con.com
WEB SERVICES
WEBMASTER
ROBERT DIAMOND robert@sys-con.com
WEB DESIGNERS
STEPHEN KILMURRAY stephen@sys-con.com
CHRISTOPHER CROCE chris@sys-con.com
CATALIN STANCESCU catalin@sys-con.com
ACCOUNTING
ASSISTANT CONTROLLER
JUDITH CALNAN judith@sys-con.com
ACCOUNTS RECEIVABLE/COLLECTIONS SUPERVISOR
KERRI VON ACHEN kerri@sys-con.com
ACCOUNTS PAYABLE
JOAN LAROSE joan@sys-con.com
ACCOUNTING CLERK
BETTY WHITE betty@sys-con.com
SYS-CON EVENTS
VICE PRESIDENT, SYS-CON EVENTS
CATHY WALTERS cathyw@sys-con.com
CONFERENCE MANAGER
MICHAEL LYNCH mike@sys-con.com
SALES EXECUTIVES, EXHIBITS
MICHAEL PESICK michael@sys-con.com
RICHARD ANDERSON richard@sys-con.com
CUSTOMER RELATIONS
CUSTOMER SERVICE MANAGER
ANTHONY D. SPITZER tony@sys-con.com
CUSTOMER SERVICE REPRESENTATIVE
MARGIE DOWNS margie@sys-con.com
JDJ STORE MANAGER
RACHEL MCGOURAN rachel@sys-con.com

compuware
compuware.com

Using Notification Servers in J2EE Applications

Free up your development time



WRITTEN BY
SUNIL MAKHLANI

For the past few years the JavaMail API has been used to implement notification services in J2EE applications. E-mails were an easy way to notify end users of business events in an application.

With the emergence of new notification channels (such as WAP phones, instant messaging applications, and SMS pagers), sending notifications has become more complicated. Now applications have to support an ever-changing set of communication channels that end users would like to be notified about. Each of these channels has a separate API that must be used to communicate with it, so a considerable amount of time is needed by developers to code these APIs into their applications. In addition, once developers have finished building the communication mechanisms, they will also need a way to determine where to contact an end user at what time (should an e-mail be sent to the user's PC, should an SMS page be sent to the user's cell phone, or both?).

In this article, I describe how a Notification Server solves the problems associated with sending out notifications in a multichannel environment. It abstracts communication code out of J2EE applications and lets end users manage the devices with which they would like to be contacted. Application developers will need to learn only how to communicate with the Notification Server, freeing up their time to work on more important tasks.

JavaMail has been used for the past few years as a standard way of notifying users of business events in Java applications. To enable business notifications, all a developer had to do was learn the JavaMail API and build the notification code into his or her application. Since e-mail was the only delivery channel available to an end user, the Java code needed to implement notifications was fairly straightforward. Listing 1 is an example of what you might use to send out a business event (an order status mes-

sage) to an end user.

As you can see, the application developer needs to keep track of a number of pieces of information in order to send an event to the end user – the e-mail server to send the e-mail, the sender address, the receiver e-mail address, and the presentation format of the e-mail itself. In this example, the message was sent as HTML. The application developer also needs to build extensive exception handling into this piece of code – e.g., should the message be resent if an exception is thrown while sending it?

The emergence of new communication channels (e.g., SMS, instant messaging, voice browsers) makes the application developer's task even more difficult; in addition to managing all the aforementioned pieces of information needed to send an e-mail, he or she also needs to manage the information for all the communication channels. The developer will also need to learn the different Java APIs needed to communicate with these channels (e.g., an SMS Java API, an instant messaging Java API). As you can surmise, application developers will need a considerable amount of time to integrate new channels into their application. If some of these notification tasks were abstracted into a single service, development time would decrease significantly.

Body

When sending notifications to different channels, device-specific considerations must be kept in mind. A simple "transcoding" solution, such as cramming an HTML-based e-mail into a two-line display on a cell phone, would not work well. Instead, a more useful approach would be to select content for

a particular channel. Let's take the case of sending a news alert to a user. Table 1 lists what the optimal content for each channel might be.

As J2EE developers have learned, separating business logic from presentation has many advantages. As a result, the Model-View-Controller (MVC) design pattern has been widely adopted by the development community to build J2EE Web applications. Similarly, when sending notifications to multiple devices, separating the notification content from the presentation layer is essential. Since each device has its own presentation content (e.g., VoiceXML for voice notifications), separating the presentation layer is important since the majority of Java developers are not experts in multichannel content.

By using XML and XSL, you can separate business content into an XML Schema, and separate presentation content into XSL stylesheets. The XML Schema stores information such as the event target user, the event subject, and the event body. The device-specific XSL stylesheets store presentation-specific content. Listing 2 illustrates a sample XML Schema containing information about a business event. Elements such as message, description, and detail are clearly defined so they can be easily formatted. Listing 3 illustrates a sample XSL stylesheet that will be used to transform the event XML document into an SMS message. Since screen size is limited on most devices that can receive SMS messages, only the message itself is displayed.

Now that I have explained how you would separate event content from presentation, you need a way to store all the various communication channels an end user would use. In addition, end

rational
www.rational.com

infrag
infragistics.com

gistics

TARGETED CHANNEL	CONTENT TO BE SENT
E-mail	An HTML forwarded e-mail, with the full body of the story and associated images
Instant messaging	The first few lines of the story, with a link to open the full body of the story in a Web browser
SMS over a cell phone	The headline of the story, with a link to open the full body of the story in a WAP browser

TABLE 1 Optimal content for a news alert

CLASS	DESCRIPTION
Event	Represents some event that has occurred. Each recipient of this event will be notified via the devices defined in his or her profile. Both the Event and EventChild extend the interface XMLTags, an interface that defines constant fields for different XML elements used within the Notification Server.
EventChild	Represents a child element of the Event whose content is not a simple datatype, such as a string or integer. An attachment or link is an EventChild.
Attachment	Abstract class representing a notification attachment.
Transformation	The base class in performing XML translations
EventQueueItem	Represents an item that will be placed in an EventQueue. The item will contain an event, which will be stored in the queue until the item is successfully delivered.

TABLE 2 Notification Server classes

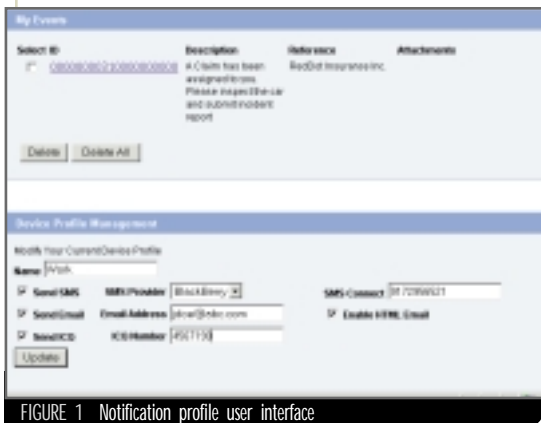


FIGURE 1 Notification profile user interface

users would also want a way to modify when they would be contacted on each specific channel. Instead of maintaining this information in your application, why not have end users do this? A Notification Server should have a Web-based application that end users can use to store profile information about the various channels (i.e., what is my e-mail address? what is my SMS number? what is my instant messaging name? etc.), as well as the times they would be notified on each channel (e-mail from 9 to 5, SMS from 5 to 10, voice notifications from 10 to 12). Figure 1 illustrates a possible front end for a user profile data entry.

AUTHOR BIO

Sunil Makhijani is a senior software engineer at Cysive, Inc., where he works in sales support for the Cymbio Interaction Server. He's also a contributing author to the BEA WebLogic Server Bible published by Hungry Minds Publishing.

Now that we've defined all the functionality a Notification Server should provide, let's look at it from an architectural perspective (see Figure 2). Both users and applications can connect to the Notification Server to send notifications out, which the server would format using XSLT stylesheets and forward to the various delivery servers (only two are mentioned in this diagram: an SMS Gateway and a SMTP Server). Users would also have the ability to set up profile information and view detailed information on alerts that have been sent to them. Finally, the Notification Server will persist all of its data to a persistent data store.

As you can see, it's easy to add new delivery mechanisms without making many fundamental changes. To add fax notifications, all you'd have to do is establish connectivity between the Notification Server and the fax delivery mechanism (a fax server), and then add an XSL stylesheet to perform a transformation of the alert data into a format suitable for a fax machine.

Now that we have a high-level view of what the Notification Server looks like, let's look at how it would be implemented in Java. Figure 3 shows part of a high-level class diagram of the core of

the Notification Server. Table 2 provides descriptions of some of the classes.

Referring back to Listing 1, let's see how Notification Servers can simplify an application developer's coding tasks. Instead of using the JavaMail API, we'll use the API provided with an off-the-shelf Notification Server. The following code snippet would accomplish the same task in less than half the amount of code shown in Listing 1.

```
Event event = new Event(new
    PortalID("jdoe"), "Order Status",
    "Your order is complete");
EventToolkit etoolkit =
    EventToolkit.getInstance()
    .createEvent(event);
```

In the first line of the snippet, an event is created using an Event object, which takes in the user, event header, and event body as parameters of the constructor. In the second line, the event is sent to the Notification Server using the EventToolkit object. Using a Notification Server, all the application developer must know is the username to send the event to. The Notification Server manages all the other information that the developer needed to know in Listing 1 (the e-mail server to connect to, the sender and receiver e-mail addresses, and the HTML presentation formatting). In addition, this piece of code will enable the developer to send SMS, voice, and instant messaging alerts; the developer doesn't have to learn any additional APIs.

Figure 4 shows what's going on behind the scenes of the call. A SAX parsing handler (EventHandler) is created to parse in the XML-based event. Once parsing is successful, an EventQueueItem (containing the Event) is added to the event queue. The Notification Server will then send out notifications from the EventQueue, based on the user profile associated with the event.

Summary

End users employ a number of channels to obtain information - e-mail, SMS, instant messaging, and voice notifications, to name a few. Although the JavaMail API was useful in the past, Notification Servers provide the most logical way to send out notifications to a variety of channels. By abstracting communication APIs (such as JavaMail) out of your code and into the Notification Server, your development time will be freed up for more important tasks. ●

smakhijani@cysive.com

oracle
www.oracle.com

Listing 1

```
// Set your mail server properties here
Properties props=new Properties();
props.put("mail.host", "mailserver.somewhere.com");

// Get a JavaMail session using the mail server
Session session =
Session.getDefaultInstance(props, null);

// Create the message
Message message = new MimeMessage(session);

// Fill the message
message.setSubject("Claim Assigned");
message.setFrom(
new InternetAddress("from@someone.com"));
message.addRecipient(Message.RecipientType.TO,
new InternetAddress("jdoe@someone.com"));
```

```
String content = "<H1> Please inspect the car and
submit incident report 024199 <H1>";

message.setContent( content,"text/html");

// Send the message
Transport.send(msg);
```

Listing 2

```
<event>
<recipient domain="cysive.com">jdoe</recipient>
<message>Claim Assigned to you</message>
<reference name="RedDot Insurance Inc."/>
<description>Claim Assigned</description>

<detail>
<name>todayDate</name>
<value>05/24/2002</value>
</detail>

<detail>
<name>Reason</name>
<value>Please determine repair costs</value>
</detail>

<detail>
<name>Claim #</name>
<value>024199</value>
</detail>
</event>
```

Listing 3

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform" >
<xsl:output method="text" version="1.0"
encoding="UTF-8" indent="yes"/>
<xsl:template match="/">
<xsl:for-each select="event">
<xsl:value-of select="message"/>
</xsl:for-each>
</xsl:template>
</xsl:stylesheet>
```

Download the Code!
www.java-developers-journal.com

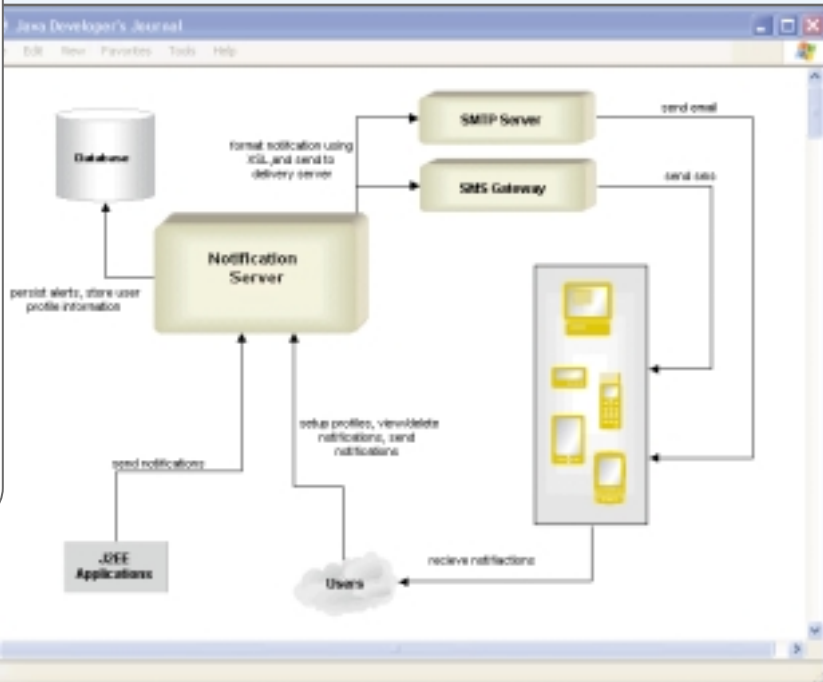


FIGURE 2 Notification architecture

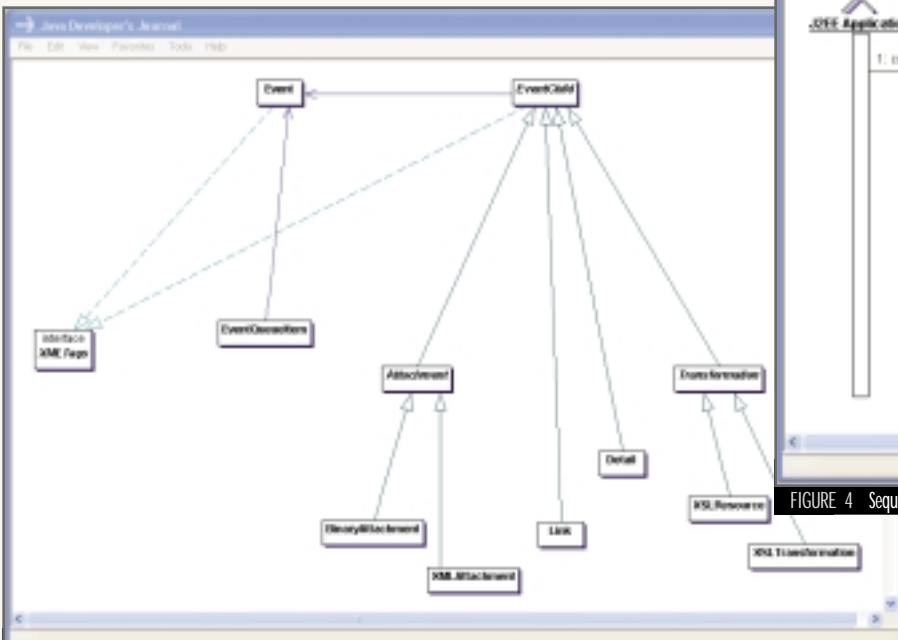


FIGURE 3 High-level class diagram of the core of the Notification Server

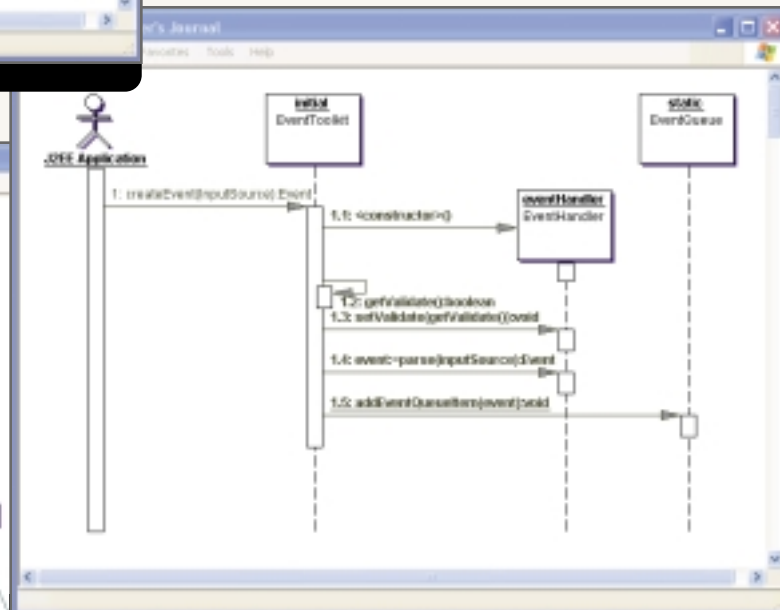


FIGURE 4 Sequence diagram

sitraka
www.sitraka.com



Making Your EJBs

written by John Musser
& Bernhard Lenz

Polymorphic

Inheritance and polymorphism are two of the most fundamental concepts in the object-oriented design world. They are used extensively in all Java applications, except J2EE apps using EJBs. Oh sure, developers implement various bean, remote, and home interfaces but often that's it. No significant class hierarchies. No polymorphic beans. Occasional reuse. Why is that?



In this article we'll first review why this is the norm in EJB-based applications and then demonstrate techniques you can use to reclaim some of these goals within the constraints imposed by the EJB standards. (We are assuming you're familiar with the basics of EJBs.) The crux of this article focuses on an application that takes the classic object-oriented design example of a hierarchy of vehicle types and translates it into polymorphic EJBs. The aim is to combine some advantages of inheritance such as code reuse and polymorphism with the advantages of EJBs such as distributed transactional components.

While this is not a universally applicable technique, you should by the end see how there's a suitable subset of problems that can be aptly solved with this approach. We'll also review interesting Java Naming and Directory Interfaces (JNDI) and other coding techniques you may find useful. A complete set of working examples (EJBs, client, descriptors, and scripts) are available on the **JDJ** Web site (www.sys-con.com/java/sourcecec.cfm) and have been tested under WebLogic 6.1 and JBoss 2.4.

Implementation Inheritance and Polymorphism

In class hierarchies you often want to treat an object instance as its base type rather than the specific specialized subtype that it is. This is a fundamental element of abstraction and is often manifest through polymorphic behavior – you define a method that expects a graphics object of type `Shape` on which you'll invoke the `draw` method but aren't concerned at runtime if this `Shape` is a `Circle`, `Square`, or `Polygon` subtype. You see this daily in Java development.

To achieve this behavioral characteristic there are two classic approaches: inheritance or interfaces. In the inheritance model there would be a base `Shape` class, likely defined as abstract, from which a set of subtypes would be derived. The base defines a set of methods and attributes and probably implements some shared behavior used by all subclasses. This implementation inheritance is typically an "is-a" approach: a `Circle` is a `Shape`.

altoweb
www.altoweb.com

In the interface model an interface such as shape would be defined, and any class that wants to “act-as-a” Shape would implement the methods in this interface (although for “shapes” the typical design is “is-a”). This interface would define one or more methods that any implementer is required to define. Also note that interfaces themselves can form hierarchies and thereby achieve interface inheritance. Again, you see this throughout the various JDKs and your own code.

The key tradeoff between these approaches is often seen through a design model that differentiates between these “is-a” and “acts-as-a” behaviors, as well as the fact that Java supports only single inheritance and interfaces provide a mechanism that allows objects to fulfill multiple roles. One of the benefits of implementation inheritance is code reuse in which common code exists or is refactored into base classes and shared by all derived types. In interface inheritance this is possible only through other design techniques such as containment and delegation. (And it’s fair to say that there isn’t always a right or wrong approach to choose – a healthy ongoing debate exists as to the design tradeoffs between the imple-

mentation versus interface inheritance.)

In EJBs, much of what occurs along these lines uses the interface-based approach. Many of the key APIs in the javax.ejb package are interfaces that your classes must define: EJBObject, EJBHome, SessionBean, EntityBean, etc. Within most EJBs the interactions with other objects typically occur through containment, delegation, and other “uses” relationships rather than through inheritance. This is reflected in most of the common EJB design patterns: session facade, value objects, and business delegates. Let’s look into why.

Vehicle Hierarchy Example

The classic introductory example of inheritance and polymorphism – a hierarchy of vehicle types – typically looks like Figure 1.

In the Vehicle base class we want to define a set of methods and attributes that all vehicles will support. Subsequently, at runtime we want to be able to treat instances of the vehicle hierarchy equally so we can invoke the startEngine() without concern for whether the exact subtype being used is a Car, Truck, or other derived type. In addition, the base class will implement some of these methods so they need not be redefined for each subtype, and some common attributes, such as fuelLevel, will be defined in the base class as well. This is all fairly basic.

In the world of EJB this isn’t nearly so simple. With EJBs a component is not one object, it’s three: the bean, its remote interface, and its home factory. It’s these three parts that cause nearly all the complications. It’s only by virtue of the Java language that each of these parts may be base or derived types, but the EJB as a whole is not “inheritable.” (As a matter of fact, the final EJB 2.0 specification in Appendix D.4 states, “The current EJB specification does not specify the concept of component inheritance. There are complex issues that would have to be addressed in order to define component inheritance.” But it does go on to say, “however, the bean provider can take advantage of the Java language support for inheritance” using interface inheritance and implementation inheritance – both of which we do here.)

Figure 2 shows the bare-bones vehicle hierarchy and the core EJB design that we’ll discuss later.

See how quickly the boxes multiply. Yet from an EJB perspective this is quite a simple picture. On the client side (for the home and remote interfaces) there’s no extra inheritance here; it’s just the bare minimum that’s required for any EJB. On the bean side there’s just a small hierarchy of those two derived types inheriting from a base (this time VehicleEJB), which in turn implements the standard SessionBean interface.

We’ll continue with our example in more detail later, but it’s instructive to first look at how our simple pieces in Figure 2 fit into the final bigger picture (see Figure 3).

As you can see, this is more complex still. What this figure demonstrates is the greater whole into which all EJBs fit. At the top are the base interfaces, such as java.rmi.Remote and java.io.Serializable, while at the bottom are the actual container-generated final implementation classes. (Note that each server is free to define and construct these actual implementations as it sees fit – for example, JBoss dynamically constructs implementations whereas WebLogic builds these when the EJBC compiler is run.)

Because of the potential complications in utilizing inheritance with EJBs, we’ve chosen a relatively simple design pattern for our approach. The key is that we perform inheritance only on the bean implementation side, expose a common remote interface, and use JNDI lookups to grab the appropriate binding at runtime.

To focus on polymorphism and object behavior, our design centers on stateful session beans (and is equally applicable to stateless session beans). With entity beans there are addition-

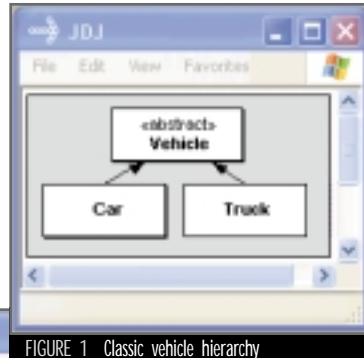


FIGURE 1 Classic vehicle hierarchy

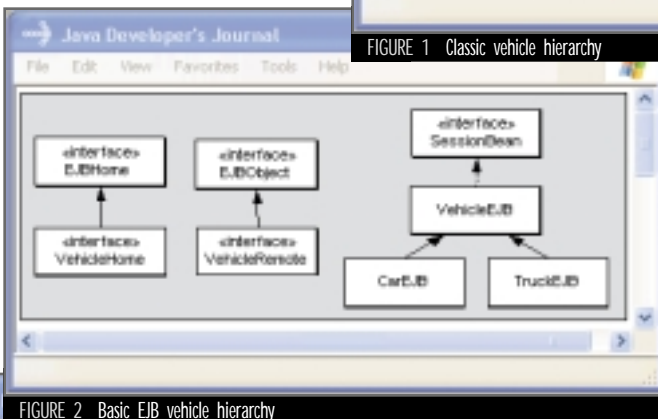


FIGURE 2 Basic EJB vehicle hierarchy

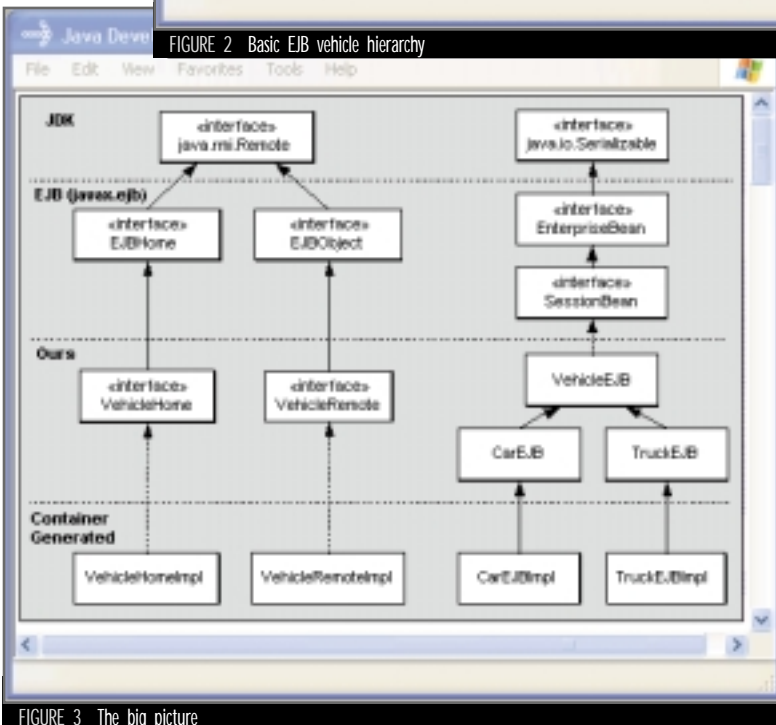


FIGURE 3 The big picture

precise
www.precise.com

al complications, such as returning differing primary keys, which we'll discuss in more detail later. As for message-driven beans, because they have no homes or remotes and a single asynchronous onMessage() point of entry, our model doesn't really apply. (As a side note, IBM, in their VisualAge for Java product, provides extensions for EJB inheritance, but because these are proprietary we don't cover them here.)

The Code

Our vehicle example consists of a single client application with the following:

1. A main() method taking arguments to specify vehicle type, color, etc.
2. A buyVehicle() method that obtains home and remote references to an EJB of the appropriate runtime type (a Car or Truck)
3. A testDrive() method to drive the generic vehicle

The first thing the client's main() method does, after validating the command line, is obtain the appropriate JNDI context. This context can be specified on the command line or use default values (we go into more background on JNDI shortly).

To connect to the JNDI tree, a context has to be created with an optional Hashtable as a parameter. The Hashtable values specify the provider's JNDI context factory as well as the URL to the JNDI tree. The code below shows how this is retrieved for JBoss (note that WebLogic's JNDI connection factory is a class called "weblogic.jndi.WLInitialContextFactory" and the default URL is "t3://localhost:7001").

```
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY,
        "org.jnp.interfaces.NamingContextFactory ");
env.put(Context.PROVIDER_URL,
        "jnp://localhost:1099");
Context context = new
        InitialContext(env);
```

Leveraging JNDI

With that context in hand, the client can call its own buyVehicle() method to get a VehicleRemote based on the command-line options. It's here that things get a little interesting.

First of all, the EJB specification requires a session bean to define an EJBHome object that contains factory-like create methods to construct the bean. In our case, this is a simple two-argument create() that takes the vehicle color and an air-conditioning flag as arguments and returns a VehicleRemote reference. The remote interface contains our shared "business" methods: startEngine(), accelerate(), brake(), and getFuelLevel(). All vehicle implementations use the same home and remote interface.

You might now ask: How can I access a specific subtype vehicle implementation on a J2EE server when the home and remote interfaces are the same for all vehicles?

This happens with the magic of JNDI. Within an EJB container, and a J2EE server more generally, the required JNDI

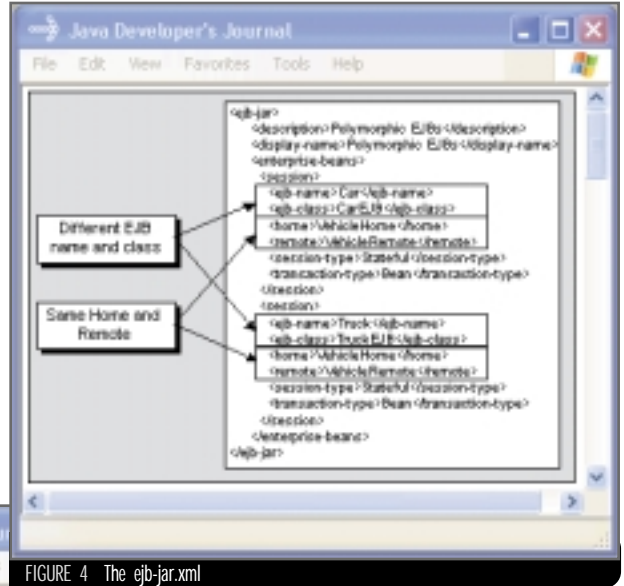


FIGURE 4 The ejb-jar.xml

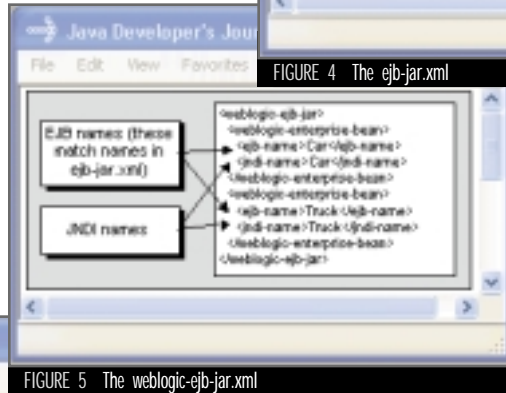


FIGURE 5 The weblogic-ejb-jar.xml

services provide a mechanism to store object instances within a globally accessible namespace (a JNDI tree). These objects can then be remotely retrieved using the JNDI name as a key. In addition, the EJB specs require application servers to publish deployed EJB home interfaces using JNDI names as defined in each EJB's deployment descriptor.

This means we can use JNDI as a level of indirection. First, each concrete vehicle implementation gets bound to the JNDI tree under its own unique JNDI name (to be more accurate, it's the home interface that's looked up by name). This VehicleHome object is then used to create a VehicleRemote reference that in turn instantiates and manages (with the assistance of the container) the designated EJB instance. It's this association between a home/remote pair and an EJB class, which is made in the deployment descriptor, that allows us to use the same home and remote interfaces but create different subtypes on the server. If this sounds a bit confusing, the next section may help clarify things.

The mechanism for this binding is specified in the following two deployment descriptors: the generic ejb-jar.xml used to describe the session beans (see Figure 4) and the vendor-specific descriptor used to assign a JNDI name to each EJB - jboss.xml or weblogic-ejb-jar.xml for JBoss and WebLogic, respectively (see Figure 5). (Note that in this example we've

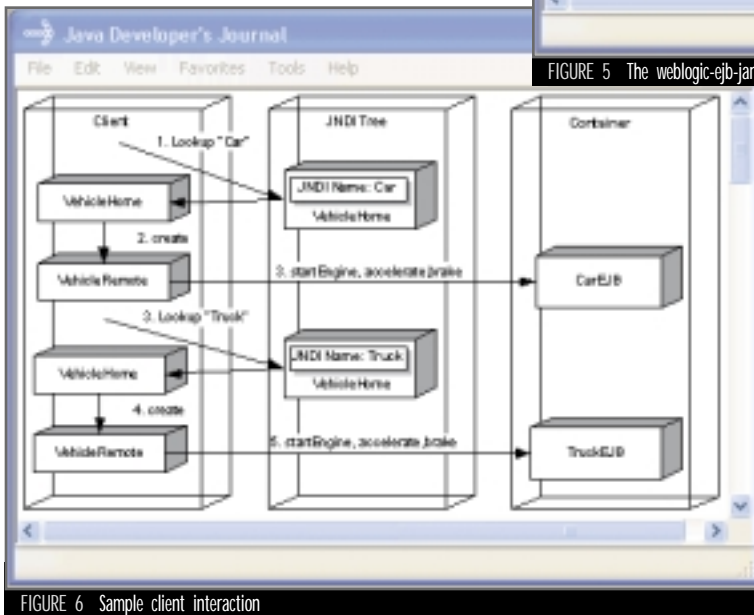


FIGURE 6 Sample client interaction

canoo
www.canoo.com

AUTHOR BIOS

John Musser is a lead architect on an e-logistics system and teaches software development at Columbia University. Over the past 20 years he has built software ranging from Wall Street trading systems to games for Electronic Arts.

Bernhard Lenz is a senior consultant with over eight years of experience in distributed, component-oriented, and multitiered systems. His clients include a variety of major Wall Street firms in insurance, banking, and brokerage. He is currently working with John Musser on a large-scale J2EE e-logistics system.

chosen to assign the beans JNDI names that are the same as their EJB names in these descriptor files – this is not required, we could have chosen differing JNDI names without breaking the rules of J2EE or our polymorphism.)

In our example, the result is that we can use the exact same code on the client side, but at runtime can construct and interact with different EJB subtypes by simply changing the JNDI name we look up.

Here's an example of the code that does this using the context we obtained earlier. First a VehicleHome is obtained from JNDI based on the binding name specified:

```
VehicleHome vehicleHome =
    (VehicleHome) context.lookup("Car");
```

Now we can create a specific vehicle (a CarEJB) using the home interface's create method that takes two arguments, a color and an air-conditioning boolean flag:

```
VehicleRemote vehicle = vehicleHome.create("silver",
    false);
```

Finally we get to drive: we start the vehicle, get up to speed, slow down for a curve, and then turn:

```
vehicle.startEngine();
vehicle.accelerate(55);
vehicle.brake(30);
vehicle.turnSteeringWheel(angle);
```

This block of code can work equally well if its remote reference is to a CarEJB or a TruckEJB.

Figure 6 shows a similar example in which a client first does a JNDI lookup on a "Car", gets the home, creates the remote, and then drives the car. It then does a lookup on "Truck" and can perform the identical steps.

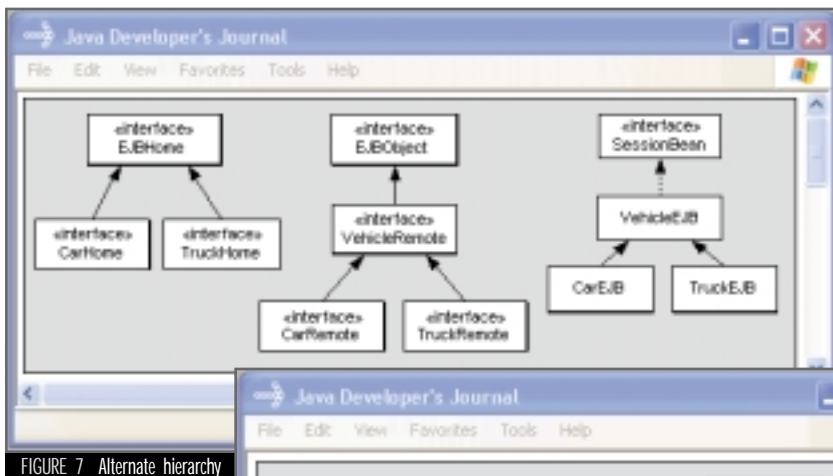


FIGURE 7 Alternate hierarchy

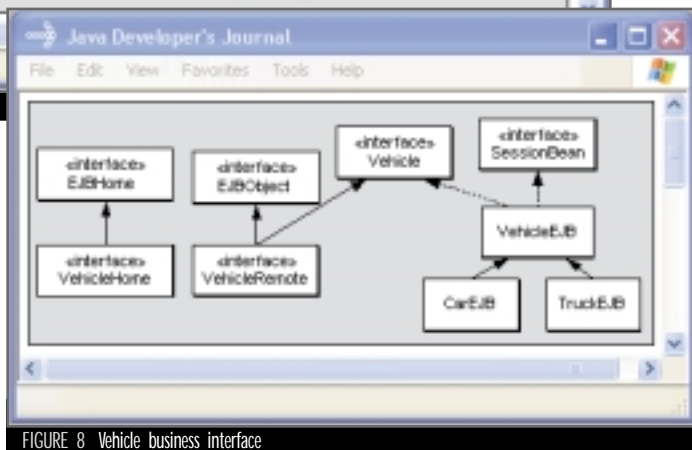


FIGURE 8 Vehicle business interface

Issues and Alternatives

As we've shown, our vehicle example capitalizes on some of the inheritance wiggle room left in the EJB specifications. However, there are additional constraints defined by the specifications that hinder what otherwise might seem viable approaches to using inheritance.

For example, it might seem that creating a hierarchy of home interfaces that parallels the beans would be nice. But the create() method of an EJBHome poses an interesting problem. In particular, the Java language does not allow overloading method signatures based on return type. Why does this matter? Because it means you can't do this:

```
public BaseHome extends EJBHome {
    public BaseRemote ejbCreate() throws RemoteException;
}

public DerivedHome extends BaseHome {
    public DerivedRemote ejbCreate() throws RemoteException;
}
```

In addition, the ejbCreate() method of bean-managed entity beans must return a primary key and derived classes must return that same type. This is true for single object finder methods as well. With finders, whether returning single objects or Collections, it would be nice if, for a hierarchy of EJBs, the base class finder could return the right subtype(s) for the given search criteria, but unfortunately this can't happen automatically with the current standard. Which in the end is why home interface inheritance is particularly problematic and why our examples avoid it – too messy.

Not to say that there aren't other EJB inheritance alternatives to consider.

A more elaborate approach is to use interface inheritance on the client side, where each derived type on the bean side gets a corresponding derived remote interface on the client side (see Figure 7).

The advantage to this approach is that it allows the client to access methods defined only in the derived types, not just those in the base Vehicle class. For example, if TruckEJB adds an unhookTrailer() method, that can be added to the TruckRemote interface. The disadvantage is that the client, at some point in the code, must now explicitly differentiate between derived types. It can no longer just pass in a different JNDI name and get a VehicleHome; it must create distinct home types (remember, we can't have a single home create different remote types). Still, at most points the client code can benefit by interacting with the more general VehicleRemote objects anywhere specific subtype methods are not needed.

(Keep in mind that in our primary example, the base VehicleEJB also adds a bit of value by fulfilling the role of an adapter pattern by providing default implementations of the EJB life-cycle methods such as ejbRemove(), ejbPassivate(), and setSessionContext() in which we cache the context in a protected member variable.)

Another technique, one that isn't so much an alternative as it is a minor modification to the first design, is to define a business interface called Vehicle and have both VehicleRemote and VehicleEJB implement this interface (see Figure 8). This provides a degree of consistency and explicit signature enforcement across both sides.

In the end, if and how you choose to use polymorphism and inheritance with your EJBs will naturally vary according to your needs. Over time, the EJB specifications will inevitably evolve and the ways in which this can be done will evolve along with them. ☛

jrmusser@hotmail.com & blenz@bermark.com

ibm
www.ibm.com

J2EE Frameworks — Extend and Submit

Maximizing productivity across
your organization



WRITTEN BY
TED FARRELL

As the Java 2 Enterprise Edition (J2EE) platform continues to grow and gain ground in corporations and the battle between Java and Microsoft .NET intensifies, more and more companies are looking for help in building their J2EE applications.

With the large number of standard technologies available, all being exposed at the programming interface or metadata level, making sense of what, when, and how to use these technologies is becoming harder.

This problem opens the door for J2EE frameworks – tools and applications that manage much of the complexities of the underlying J2EE data, code, and deployment and allow the user to work in a much more efficient and coherent manner. A framework centralizes different common design-time and runtime resources, allowing developers to concentrate on the unique business logic of the application, and avoid having to build the underlying infrastructure to support the applications.

This can lead to bigger productivity gains for developers, as well as more flexibility for corporations as many of the frameworks don't require in-depth knowledge of a particular programming language or technology, thus creating more opportunities for a diverse group of developers to contribute to building the application. The catch is that frameworks must be able to do this without locking the users into proprietary tools and data, which isn't easy.

Current Frameworks

Frameworks aren't new. Different types of frameworks have been in existence for over a decade. They're designed to help an organization build and deploy an application more quickly and efficiently. Some come with a packaged application for use in customizing that application, while others are more

generic in nature, allowing users to build custom applications.

One of the problems with traditional frameworks is the lock-in. With every proprietary framework comes tool, data, and vendor lock-in to that framework. Committing to using one of these frameworks means committing to the framework vendor and its data format for the life of the application. With the lack of industry standards and ever-growing time-to-market pressures, many corporations have bought into these frameworks to help build their applications. These frameworks help most companies build their applications quicker. PowerBuilder and Forte are good examples of these types of proprietary frameworks.

Problems don't surface until a change in the industry, technology, or application requirements forces the company to make changes in their application, and often in the technology stack supporting the application. The Internet was one of the biggest catapults to this crossroad. To reach more users and eliminate costly software distribution and upgrades, developers began fleeing client/server architectures and heading for the Web, with simpler, browser-based user interfaces for their applications.

At this point, the company is at the mercy of the framework provider and the technology and direction they're promoting. If the framework can't support the new requirements or the framework provider chooses not to support these requirements, the company is stuck with an application that requires a massive port or migration to continue

evolving, or in more extreme cases, a complete rewrite onto a new technology stack.

This dilemma is not unique to users of proprietary frameworks. Companies that choose to build the underlying application infrastructure themselves can run into similar problems. The lack of industry standards in building applications lends itself to this inevitable predicament.

J2EE

J2EE is designed to solve this problem. With the support of every major software vendor in the market except one, J2EE has quickly become a robust, scalable solution for building enterprise applications. With over 30 vendors shipping J2EE-compliant runtimes and thousands of customers building and deploying J2EE applications on those runtimes, J2EE has proven itself as the industry-standard technology for building enterprise applications.

To prevent the technology dead end that was mentioned earlier, J2EE remains a living thing. More than 300 members of the Java Community Process (JCP) are continuously working to grow and improve the standard. While no one company can come up with all the answers, the JCP ensures that a multitude of diverse knowledge and experience goes into building the Java 2 standards. As the industry grows and changes, so do the J2EE and associated standards. Each addition to the specifications is an extension to the architecture, thus allowing customers building applications on J2EE to evolve them as

borland
www.borland.com

technology evolves, as opposed to migrating, integrating, or rewriting them.

One of the major benefits of J2EE is how the architecture separates the different components and functionality of the application. The design promotes the separation of functionality, such as keeping the business data, the business logic, and the presentation layers separate from each other. This allows organizations to build more flexible applications that are able to change as the market and industry change, as well as much easier to maintain and support. The downside to this approach is that this architecture becomes more complicated to manage. The application is broken up into different pieces, and developers and businesspeople working on it need to be aware of those different pieces and how they all work together. It's a trade-off between a flexible, scalable architecture and development complexity.

This puts the pressure on the tools and development environments to hide these complexities and allow developers to focus on the task of building the application and not be bogged down by all the details of the underlying architecture. Java Integrated Development Environments (IDEs) have made significant progress over the last year in providing a

toolset that greatly reduces some of the complexities of Java, J2EE, and Web services development and deployment.

While in the past the Java community has been criticized for not having tools with the depth or integration of Microsoft's tools, that gap has been greatly reduced over the past 12 months. Although a large number of developers gladly welcome the new generation of IDEs and are receiving immediate benefits from them, others still find them either too heavy or still too technical for their needs.

This brings up the question: Can one tool solve the problems of all the developers in a company? The answer is "No." This answer is something that software providers have avoided admitting for years. A single tool or product that can appeal to every developer, businessperson, and analyst in your company does not exist, and never will. The good news is that the J2EE architecture provides the platform, infrastructure, and process that are able to solve this problem. The industry just needs to realize it and work together to accomplish it.

J2EE Frameworks

Frameworks can be a powerful mechanism to abstract the complexities of the infrastructure away from the logic of the application. This can lead to larger productivity gains and a more maintainable application. Another advantage is that users don't have to completely understand the underlying technology in order to contribute to the building of the application. The level of abstraction will vary between frameworks depending on the target audience and intended usage.

Currently, several products on the market claim to be J2EE frameworks. They accomplish this by providing a proprietary framework that eventually either generates code that runs on a J2EE application server, or generates metadata that's processed by components running on a J2EE application server. In either case, users are stuck with proprietary metadata describing their application. J2EE is not about standardizing implementations, but rather standardizing to the implementations in the form of the code and metadata consumed by the implementations. Having your data in a proprietary format that requires a proprietary tool to manipulate does not give you the portability or freedom of choice promised by the J2EE architecture.

These frameworks behave this way because it's easier. It's easier to

work with concise, complete, and proprietary data than with the complex and modular data of the J2EE architecture. It's easier to invent things yourself than to work with the JCP to make them part of the standards. It's easier to claim the standards are incomplete than it is to work to extend them to meet your needs. It's easier to regenerate code from metadata than to build up metadata from code. The result of these choices, however, is a lot of proprietary data formats that lock companies into a vendor and toolset, and, ironically, have them end up with the opposite result of what they were trying to accomplish: to build their application on open standards.

Alternatively, the definition of a J2EE framework should be a set of design-time tools and, optionally, runtime components that ease the development, management, and deployment of J2EE applications by directly manipulating J2EE code and metadata. The features of a J2EE framework are not as important as the data that it manipulates. There will be multiple frameworks that target different types of users, performing different tasks with different skill sets. By this definition, a Java IDE can be considered a framework. Current Java IDEs can help in the creation, management, and deployment of J2EE applications, all while editing standard Java code and J2EE XML metadata. A user can then take that code and metadata and manipulate it with any other Java IDE, regardless of the vendor.

If J2EE frameworks work off a standard set of code and metadata, companies can mix-and-match the types of tools and frameworks that they use throughout the development organization and process. Each framework can have a particular set of features targeted at different types of users and still manipulate the same underlying data as all the rest of the frameworks. This would include frameworks from different vendors as well.

For example, there's a J2EE framework that allows the user to pick a database table and create an object so he or she can use that table information in the program. Behind the scenes, the framework might be creating an entity EJB with its multiple Java files and XML deployment files. Then another user, who does not use the same framework as the first user, is able to work on the same EJB using a favorite IDE or even a text editor. The first user may know nothing about Java or EJBs and the second user might know everything about Java and EJBs, but they can work together on the same project and the same code base. This is almost impossible with the frameworks on the market today.

north woods



J2ME



J2SE



J2EE



Home

jinfonet
www.jinfonet.com

Extend and Submit

Accomplishing the task of standardizing the code and metadata across all tools and frameworks becomes more difficult as the level of abstraction in the framework increases. The more the framework removes the user from the code and metadata and incorporates process, the more mapping the framework vendors must do to store their information in the standard formats. Because the J2EE standards are written at the code and implementation level, the data formats are not always designed to store the abstract information that would be needed by many of the frameworks. This could lead to vendors using a mix of some standard data as well as some proprietary data where the standards, didn't support a particular notation or abstraction. Although this is better than a totally proprietary solution, without the right commitment to the standards, this still lends itself to a certain level of vendor, tool, and data lock-in.

The resolution to this is to extend and submit. When framework vendors come across a situation in which the current standards do not provide adequate means to store some metadata that's needed to support their framework, they should extend the current

specification, and then submit those changes back to the standards community.

The Java Community Process has been very good at turning around requests to extend the Java 2 standard in a fairly short time. (These requests are called Java Specification Requests or JSRs.) The JCP is also good at providing lots of information on these JSRs to the public during the standard-making process.

There is a chance that the requirement the framework vendor is looking for is already covered in an existing JSR that's still in process and has not yet become part of the Java 2 standards. In this case, vendors should align themselves with the JSR and ensure that what they produce will be compliant with the technology once it becomes part of the standard. In other situations, vendors' requirements may not be covered in an existing specification or JSR. In this case, vendors should submit a JSR to the JCP with their request while developing their framework. This path requires more discipline on the part of the framework vendors because they'll be producing a technology that the JCP will be debating and changing as it becomes part of the standard.

Throughout the JSR process, vendors' specific requirements may turn

into the industry's broader requirements, causing vendors to revisit their implementation to align with the new standards. This will result in vendors spending more time implementing their solution, which in turn will ensure that their solution is part of the standard.

A good example of this is the EJB support in J2EE. With versions 1.0 and 1.1 of the EJB specification, application server vendors needed to store more information than the specification allowed; their runtimes were more advanced than the specifications. The result was that vendors extended the EJB specification in their own proprietary manner. Then with version 2.0 of the EJB specification, those vendors were able to collaborate and extend the EJB standard to include additional information. The EJB specification was extended, and then submitted back for inclusion in the standards. The alternative to this is for vendors to continue producing proprietary metadata for their frameworks, which will lead to the fragmentation of the standards.

Industry Effort

To help stop the fragmentation of the J2EE standard, Sun Microsystems supplies a J2EE Compatibility Test Suite (CTS) to any vendor wishing to license the J2EE technology. Vendors must pass these tests in order to claim they are J2EE-compliant. As mentioned earlier, J2EE is primarily focused on runtime implementation, not design time. Therefore, all these compatibility tests are focused on runtime, not design-time, compatibility. This means that as long as vendors produce something that complies with the J2EE runtime standards, they're considered J2EE compliant, regardless of how they store the metadata that is used to create the compliant runtime code. Currently, no test can validate that the metadata a tool is producing is J2EE compliant, just the data that it deploys.

This means the framework vendors must remain disciplined when building their frameworks and adhere to using standard code and metadata. This makes the job of building these frameworks harder, especially when competing with Microsoft .NET. Microsoft has the luxury of controlling the runtime, the data, and the tools – one of the advantages of a totally proprietary solution. Therefore it's up to the J2EE vendors to continue to prove that you don't have to give up a robust, scalable, and open architecture just to get a productive set of tools. Users will be able to benefit from all the advantages J2EE has

capela
capela.com

esri

www.esri.com

over Microsoft .NET, without sacrificing efficiency, flexibility, or ease of use.

Framework Architecture

Just as having one tool cannot satisfy the needs of everyone, having several different tools, even if they're sharing the same underlying data, can also be a headache. Multiple tools can lead to more maintenance and management problems, overlapping technologies, and questions about what to use when and why. To get around this, J2EE frameworks should not be standalone technologies, but rather standalone functionality plugged into a common architecture.

Some of the Java IDE vendors have extendible architectures. This is different from an add-in API to an IDE because with extension architecture, you can build tools that have nothing to do with IDEs. The architecture provides you with common resources such as project management, navigation, file I/O, and data sharing, and lets the different extensions define the functionality. Each extension is treated as a first-class entity and can share the data and the processes of the application with the other extensions.

By building J2EE frameworks on top of a common architecture, the frameworks can take advantage of the resources already provided by other

extensions, such as compiling, code generation, parsing, debugging, etc. There's no need for the framework to build this duplicate technology when it can share it with the rest of the products.

Another advantage to using a common architecture has to do with the problem of trying to make everyone happy. For example, a user using a framework wants to cut out and edit some code directly. If the framework is a different product than the user's Java IDE, he or she needs to save all the work and then bring up the IDE, load the data, and begin working. When done, he or she needs to perform the opposite tasks to get the framework up to date with the code changes. With a common architecture, however, all the functionality is in the same product. The user would just switch to the code editor and make changes. When the user switches back to the framework (tab, panel, window, etc.), the framework is automatically notified by the architecture of the changes and updates. In this environment, the user is not bound by any one framework, but can use functionalities of different ones when needed. Since all the frameworks work off the same data, everything is kept synchronized.

Summary

Building enterprise applications can be complicated. J2EE provides a proven, scalable, and robust platform for accomplishing this task. As the technology continues to grow and companies are looking for more ways to utilize the J2EE infrastructure, J2EE frameworks will play a key role in coordinating and optimizing the resources and efforts across organizations. To ensure that this next phase of productivity tools and frameworks is successful, the J2EE community must remain disciplined and focused on preserving and growing the Java 2 standards by committing to maintaining a common, standard set of code and metadata.

Just as J2EE application vendors will continue to compete on who has the best implementation of the runtime standards, J2EE framework vendors will compete on who has the best set of features and processes to build applications using the standard metadata. With all the different types of users and requirements, the market to provide J2EE frameworks could be big. Vendors will be able to compete in this market by the features that their frameworks provide and not their proprietary data. ☛

ted.farrell@oracle.com

int
www.int.com

AUTHOR BIO

Ted Farrell is architect and director of strategy for application development tools at Oracle Corporation. He's responsible for the technical and strategic direction of Oracle's development tools products, including Oracle9i JDeveloper, Oracle's Java and XML Integrated Development Environment (IDE) for J2EE application and Web services development.



J2ME



J2SE



J2EE



Home

new atlanta
newatlanta.com



JASON BELL J2SE EDITOR

Desert Island Open-Source Disks

You may be aware of a radio program in the UK called “Desert Island Discs.” Basically, well-known people choose which records they would want if they were stuck on a desert island (I’ve yet to hear anyone say they’re taking a CD player). Something of a similar nature is happening to me at the moment, as I’m working from home (but far from stranded).

Since I didn’t have any of my normal development tools on the laptop I borrowed, I had to hunt around the Internet and download the tools I needed to get my jobs done. The tools all had one common feature – they’re all open source. So I decided to present my selection of “Desert Island Open-Source Disks.”

jEdit

(www.jedit.org)

I’m currently working with version 4 of jEdit and it’s a joy to use. In addition to being a normal source editor with syntax highlighting, it has the ability to use plug-ins. If there’s one plug-in I would advise you to download, it’s JavaStyle as it tidies up your code layout and also inserts the relevant JavaDoc comments.

Jikes

(www-124.ibm.com/developerworks/oss/jikes/)

IBM’s open-source Java compiler has more meaningful error messages than the original compiler. It also suggests where to use try/catch/finally blocks when working with code that throws exceptions. It handles all the same command-line tags as javac and is very fast at compiling.

JUnit

(www.junit.org)

If you’re using the Extreme Programming route to get software development projects done, you may have come across

JUnit already. For the rest of the world, this little utility runs unit tests on your Java code; all you have to do is create a small class to run the code. There’s a more in-depth look at JUnit in the article “Test First, Code Later” by Thomas Hammell and Robert Nettleton (*JDJ*, Vol. 7, issue 2).

Ant

(<http://jakarta.apache.org/ant/>)

This is the build tool that everyone seems to be using, so it would be silly of me to even attempt to leave it out. Personally, I’d be lost without it. As with most software that comes from the Apache Foundation, be prepared to play around with how things work and also to read some of the documentation. Once you get going though, you can’t imagine how you got on without it.

SCP/MindTerm SSH Client

(www.isnetworks.net)

MindTerm is a secure shell (SSH) client built entirely in Java. ISNetworks added a secure copy (SCP) function and made it available for download. More and more people have disabled the telnet access in favor of SSH and I’m glad to see it happening, for everyone’s peace of mind.

The following demonstrates the benefits of open source. I had a problem with the directory listing since FreeBSD was not recognizing some of the flags in the “ls” command. I fired up jEdit, found the source file, and corrected the problem. A quick compile and I added it back to the JAR file and was back at work.

There you have it – the software that kept me going. All free and with its source code so you can either improve its design or functionality or just mosey around and see how it works. The software you use will depend on who you work for, since

Desert Island Open-Source Disks

There’s a radio program in the UK called “Desert Island Discs.” Well-known people choose which records they would want if they were stuck on a desert island. So I decided to present my selection of “Desert Island Open-Source Disks.”

by Jason Bell

36

A List Tool for All Applications

How often do you copy and paste old code when developing GUIs? If I do that more than once or twice on the same code, it’s worthwhile making it a component for the benefits of reuse and highly maintainable code.

by Teresa Lau

38

Performance Tuning in Java

It’s common practice to ignore efficiency, scattering layers of unnecessary inefficiency everywhere. This article shows that it’s just as easy to write faster code without taking extra development time to do it.

by Dov Kruger

44

sometimes you don’t have any say in the matter. I try to encourage people to use the tools that suit them and that will enable them get the job done in a manner everyone is happy with.

The nice thing about open source is that there are a lot of people willing to share a lot of information. If you want to know more about open-source principles, *The Cathedral and the Bazaar* by Eric S. Raymond was helpful. Though it may be geared toward the Linux folks, it’s still a good read and gives you a bit of history, a few examples, and some other texts to ponder over (once you’ve read *JDJ* word for word, though). ☛

▼▼▼ jasonbell@sys-con.com

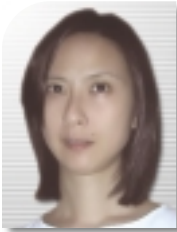
AUTHOR BIO

Jason Bell is a programmer and senior IT manager for a B2B Web portal in York, England. He has been involved in numerous Web projects over the past five years, the last two of which have been servlet-based.

macromedia
macromedia.com

A List Tool for All Applications

Designing for reuse and code maintainability



WRITTEN BY
TERESA LAU

How often do you find yourself copying and pasting old code when developing GUIs that are similar to others you've developed? Many times I find that if I do that more than once or twice on some code, it's worthwhile to make a component out of it for the benefits of reuse and highly maintainable code.

The initial overhead for creating a component is high, but it will pay off if you design it well, making it useful for other occasions. The better you design it, the more likely you'll use it in the future and the more benefits you'll get.

ListPanel and ListTabPanel are two reusable components I developed. In this article, I describe how you can use them to simplify and speed up your GUI development. I also explain how they're implemented and hope you can learn some useful skills from them.

ListPanel

ListPanel is a GUI component that allows a user to filter, sort, display, and search a list of any object in different ways. Figure 1 shows a list of Auction items that can be filtered by their status (open, closed). The list can sort, display, and search by the Auction item number or name. Using this component the developer writes only a few lines of code and some inner classes to create the list GUI, which normally involves complicated code. It's easy to understand and the code is very maintainable.

Please note that for simplicity, this component assumes that the way you display and search data is the same way you sort data. You can modify this component to make it more sophisticated and to handle the sort, display, and search differently.

Creating an Instance of ListPanel

The following shows how easy it is to create the GUI in Figure 1 using the ListPanel.

The ListPanel contains a list of Auction objects (see Listing 1 for the Auction class). (Listings 1-7 can be downloaded from www.sys-con.com/java/sourcecode.cfm.)

```
ListPanel listPane = new ListPanel();
listPane.setData(createTestData ());
listPane.addFilterSpec(new
    StatusFilter("Open" ) );
listPane.addFilterSpec(new
    StatusFilter("Closed" ) );
listPane.addSortSpec(new IDSort ());
listPane.addSortSpec(new NameSort ());
listPane.refreshGUI();
```

This code involves only five steps:

1. Create the ListPanel with the constructor.
2. Set the data (in this case, a list of all Auction objects) by setData.
3. For each FilterSpec, call addFilterSpec.
4. For each SortSpec, call addSortSpec.
5. Call refreshGUI once you've set up all the filter and sort conditions.

Creating a FilterSpec

To create a FilterSpec, extend the abstract class FilterSpec (see Listing 2) and provide implementation for the following abstract methods:

- **String getName():** Defines how the filter name is displayed in the JComboBox
- **boolean evaluate(Object o):** Defines the filter condition

To create the filter based on the Auction status use the following:

```
class StatusFilter extends FilterSpec
{
    private String status;
    public StatusFilter (String s)
    {
        status = s;
    }
    public boolean evaluate(Object o)
    {
        return (((Auction)
            o).status.equals(status));
    }
    public String getName()
```

```
{
    return status;
}
```

Creating a SortSpec

To create a SortSpec, extend the abstract class SortSpec (see Listing 2) and provide implementation for the following abstract methods:

- **String getName():** Defines how the sort name is displayed in the JComboBox
- **getSortString(Object o):** Defines how to sort, display, and search by

The code for creating the sort by ItemNo, Name of the Auction is as follows.

```
class IDSort extends SortSpec
{
    public String
        getSortString(Object o)
    {
        return ((Auction) o).itemNo;
    }
    public String getName()
    {
        return ("Item No");
    }
}
```

```
class NameSort extends SortSpec
{
    publicString
        getSortString(Object o)
    {
        return ((Auction) o).name;
    }
    public String getName()
    {
        return ("Name");
    }
}
```

Listing 1 provides the code to generate the AuctionList.



FIGURE 1 AuctionList

silverstream
silverstream.com

Another Example

The ListPanel is a flexible component. Figure 2 shows another list created on a totally different object, User. This list has three filters (active, inactive, all) and two sorts (ID, LastnameFirstname). This entire GUI is created in one routine, createListPane() shown in Listing 3.

In Listing 3, you can see that because the FilterSpec and SortSpec are all small classes, I've made them all inner classes. In situations where I have a lot of little classes used in one place, inner classes are useful to avoid proliferation of classes.

ListTabPanel

ListTabPanel contains a ListPanel and a TabPanel. It provides a structure that you can add tabs to to perform different actions on a particular selected object from the list. The implementation uses the Mediator pattern, where the ListTabPanel is a mediator that takes care of the communication between the ListPanel and the TabPanel.

Using the same Auction class I discussed earlier and the AuctionList, I create the AuctionPanel shown in Figure 3.

Creating the AuctionPanel involves three simple steps:

1. Create the Auction ListPanel defined in the previous section.
2. Create a TabPanel that contains three different tabs.
3. Call the constructor, passing it the ListPanel and the TabPanel.

```
ListPanel l = AuctionList.createListPanel();
TabPanel t = createTabPanel();
ListTabPanel p = new ListTabPanel (l, t);
```

The details for creating the TabPanel in createTabPanel follows:

```
TabPanel t = new TabPanel();
t.addTab(new AuctionTab());
t.addTab(new MinimalTab("Seller"));
t.addTab(new MinimalTab("Buyer"));
```

Here I create the TabPanel, and then add as many TabObjects as needed by calling addTab. Each TabObject defines how the object is displayed and handled in a particular tab. Let's look at how the first tab is implemented in AuctionTab.

```
public class AuctionTab extends TabObject {
    public AuctionTab() {
```

```
// ... Create the GUI here
}

public void setData(Object o) {
    // ... populate the data to the GUI
}
}
```

The AuctionTab extends TabObject, which is a JPanel. The developer usually does three things to implement this tab:

1. Creates the GUI in the constructor
2. Implements the method setData to populate the GUI with an object
3. Manipulates the object in its own way and updates it as appropriate (e.g., save to database, etc.); when manipulated, an object should call the appropriate methods notifyAdd(), notifyDelete, or notifyUpdate() to let its listener know of the change

Listing 4 provides the code for AuctionPanel.

Implementation of the ListPanel

The ListPanel component provides a GUI that can filter, sort, display, and search any kind of object. The component has no idea what kind of objects are in the list and how it should filter or sort; it's up to the developer to define this information by calling:

- setData (List data)
- addFilterSpec (FilterSpec f)
- addSortSpec (SortSpec s)

With this information from the developer, the component then handles the rest, which includes:

- Creating JComboBox to allow the user to choose the methods to sort and filter
- Filtering the data according to the filter the user selected
- Sorting the data according to the sort the user selected
- Displaying the data according to the sort the user selected
- Implementing the search according to the sort the user selected

Jakarta Common Collection

To implement the filter and search mechanism, I need some methods where I can:

- Filter from a list based on some condition and return a filtered list
- Search from a list based on some condition and return the first object that matches

While the Collections class in the java.util package does provide a search

method, binarySearch(List l, Object o, Comparator c), there's no mechanism to filter the data in the list. Furthermore, this search method requires that I sort the list before passing the list in. In addition, if I want the filter condition to be a union or intersection of two conditions, the Collections class doesn't have set theory methods that I can apply to the collections.

The Jakarta Commons Collections (<http://jakarta.apache.org/commons/index.html>) is a set of reusable components related to collections. It strives to provide some features that were left unfilled by Sun's implementations in the Collection class.

Using Jakarta Commons Collections you can search and filter a collection of data based on a Predicate using the method search and find in the CollectionsUtil class. Predicate is an abstract class with a method evaluate(Object o) that performs some Predicate that returns true or false based on the input object. In addition, the CollectionUtils class provides set theory methods like intersection and union if you want to operate the filter on more than one condition.

Implementation Details

The ListPanel has three pieces of information after the developer creates an instance of it: dataList (a list of all objects), filterList (a list of all filters), and sortList (a list of all sorts).

Using filterList and sortList, ListPanel creates combo boxes that contain lists of all available filters and sorts. When the user selects a sort or a filter, the currentFilter or currentSort object is updated, and the method refresh is called. In refresh, the data is first filtered as follows:

```
// Filter the data according to currentFilter
filteredData =
    CollectionUtils.select(data, currentFilter);
displayModel.clear();
Iterator i = filteredData.iterator();
while (i.hasNext())
    displayModel.addElement(i.next());
```

I get the filteredData by calling the select (List l, Predicate p) method in the CollectionUtils class of Apache Common Collections. You should now see why FilterSpec implements Predicate, because now all I need to do is to pass currentFilter (which is a FilterSpec) to the select (List l, Predicate p) method.

After the data is filtered, it's sorted as follows:

```
// Sort the data according to currentSort
```



FIGURE 2 User list

dice
www.dice.com

```
displayModel.sort(currentSort);
displayList.setModel (displayModel);

// Display the data according to
currentSort
displayList.setCellRenderer(current
Sort);
```

I created a class `SortableListModel` (see Listing 5) that can sort data by calling sort (`Comparable c`). The `displayModel` you saw earlier is a `SortableListModel`, so I sort this list by simply calling `displayModel.sort(currentSort)`. You should now see why `SortSpec` implements `Comparator`, because with that, all you need is to pass `currentSort` to this sort method.

Next, I display the data in the `JList` by setting the `cellRenderer`. Again you can see that `SortSpec` extends `DefaultListCellRenderer`, which means you can just pass `currentSort` to the `setCellRenderer(ListCellRenderer)` method.

Last, when the user types something in the text field and hits the search button, the method `search` does the following:

```
currentSort.setSearch(searchTF.getText
());
Object o = CollectionUtils.find
(filteredData,currentSort);
displayList.setSelectedValue(o, true);
```

Here I first set the text to be searched, then use the `CollectionUtils.find` (`List list, Predicate predicate`) method to find the first object that satisfies the `Predicate` from the list. Again, you see that `SortSpec` implements `Predicate`, which is why you can just pass `currentSort` to the `CollectionUtils.find` method. Once the object is found, select it on the `JList`.

Listing 5 provides the code for the `ListPanel`.

Implementation of ListTabPanel

To understand how the `ListTabPanel` is implemented, you first need to understand the `Mediator` pattern. `Mediator` is a pattern that promotes

loose coupling between classes. It accomplishes this by being the only class with detailed knowledge of the methods of other classes. Classes inform the `Mediator` when changes occur, and the `Mediator` then passes them on to any other classes that need to be informed.

The `ListTabPanel` (which contains the `ListPanel` and the `TabPanel`) is the `Mediator` here. The `ListPanel` and `TabPanel` don't know about each other. They notify only the `ListTabPanel` of changes in themselves, or get notified by the `ListTabPanel` when changes come in. Clearly defining the tasks of each component will make coding easy and clear. Let's now look at the tasks of each component:

Tasks of the ListPanel

- Provide a list that can filter, sort, display, and search in different ways.
- When a user selects an item, it should notify its listener (in this case the `Mediator`).
- Provide update, delete, and add methods so another object (in this case the `Mediator`) can adjust its list data.

Tasks of the TabPanel

- Provide a `setData` method for each tab so that another object (in this case the `Mediator`) can set data and populate the tab accordingly.
- When a tab calls update, delete, or add on an object, it should notify its listener (in this case the `Mediator`).

Tasks of the ListTabPanel

- When an item is selected from the `ListPanel`, this component (as a `Mediator`) gets notified and calls `setData` on the `TabPanel`.
- When the `TabPanel` updates some object, this component (as a `Mediator`) gets notified and calls update, add, or delete on the `ListPanel`.

Using PropertyListener

In my implementation, the communication between the `ListPanel` and `TabPanel` and their `Mediator` is done by `PropertyListener`. The `ListTabPanel` is a `PropertyChangeListener`. When you construct a `ListTabPanel` with a `ListPanel` and a `TabPanel`, the `ListTabPanel` will add itself as a `PropertyChangeListener` to the `ListPanel` and `TabPanel`.

```
tabPane.addPropertyChangeListener(this
);
listPane.addPropertyChangeListener
(this);
```

By doing so, when the `TabPanel` or `ListPanel` wants to notify the `ListTabPanel` of anything, it'll call `firePropertyChange` to pass the property change and the object that was changed to the `ListTabPanel`.

The `ListTabPanel` received these change events in its `propertyChange` method, and it will do the appropriate thing to notify other components if necessary.

```
public void propertyChange
(PropertyChangeEvent evt)
{
// Get notify when ListPanel
select some object
if (evt.getPropertyName().equals
(LIST_SELECTED))
// Ask TabPanel to set the
object to its screen
tabPane.getSelected().setData
(evt.getNewValue());

// Get notify when TabPanel update
some object
if (evt.getPropertyName().equals
(TAB_OBJECT_UPDATE))
// Ask ListPanel to update its
list
listPane.update(evt.getOldValue(),
evt.getOldValue());

// ... Other events
}
```

The full source of `TabPanel`, `TabObject`, can be found in Listing 6, and the full source of `ListTabPanel` can be found in Listing 7.

Conclusion

The `ListPanel` and `ListTabPanel` are two useful GUI components that are highly reusable and can promote rapid GUI development. The `ListPanel` takes care of the common features like sort, filter, search, and display on a list. The `ListTabPanel` takes care of the interaction between a list of objects and the detail panels on a specific object in that list. These two components are created so that they're applicable on any kind of object and provide a clean interface for developers. Using them, developers can concentrate on coding the details specific to their object.

More important, this article not only described how to use these components, but also their implementation. It's useful to understand the whole design process for making a component reusable and loosely coupled, resulting in clear and maintainable code. ☛

AUTHOR BIO

Teresa Lau has been an independent Java consultant for over four years, with an emphasis on financial applications. She received her MS in computer science from the University of Waterloo, and her BS in engineering from the University of California, Berkeley.

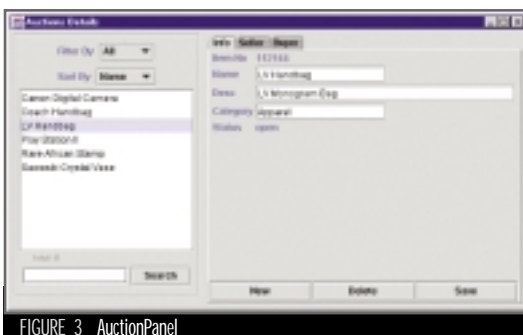


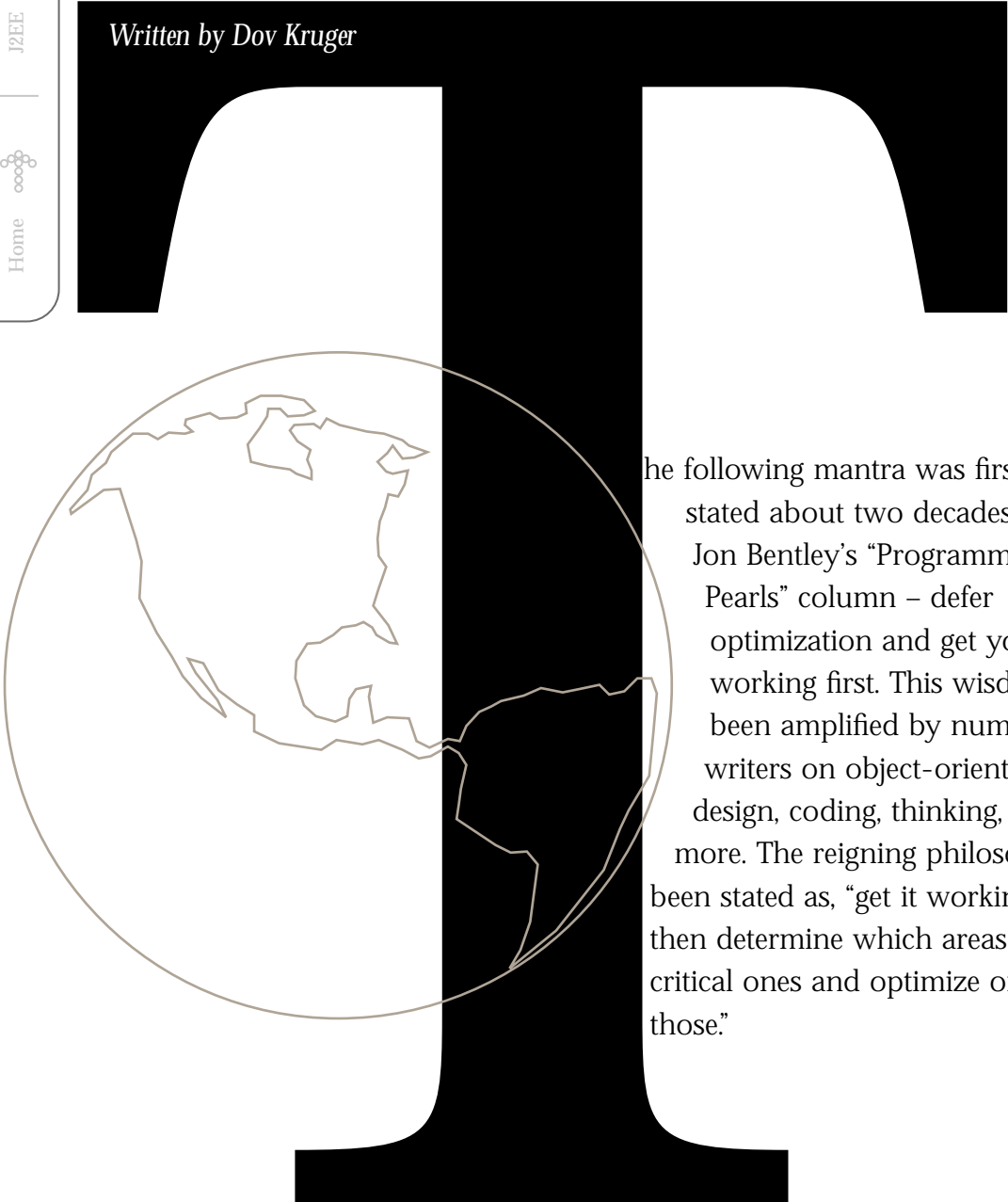
FIGURE 3 AuctionPanel

parasoft
www.parasoft.com

Performance Tuning in Java

Written by Dov Kruger

*For faster code
without a lot
of effort*



The following mantra was first stated about two decades ago in Jon Bentley's "Programming Pearls" column – defer optimization and get your code working first. This wisdom has been amplified by numerous writers on object-oriented design, coding, thinking, and more. The reigning philosophy has been stated as, "get it working first, then determine which areas are the critical ones and optimize only those."



J2ME



J2SE



J2EE



Home



Since 20% of the code is run 80% of the time, this seems like a reasonable idea. Bentley was an early advocate of the use of profiling tools that show which parts of the code are run the most, and targeting the most critical areas first. On the whole, this is good advice, but the lesson has been learned too well. Today it's common practice to ignore efficiency, scattering layers of unnecessary inefficiency everywhere without thought. This article shows that it's just as easy to write faster code without taking extra development time to do it, and teaches something about the way Java optimizes your code.

Method Calls

Object-oriented design is an organizational technique; at the individual method level, code has been written the same way for the past 40 years. The focus on organization merely breaks code down into small manageable units – classes that contain a number of (typically quite small) methods. In a language such as C++ this is not a problem at all, as the language itself has extensive control over the cost of the method calls. In fact, with the inline directive, the cost of method execution in C++ can drop to zero. In Java, however, the default method definition checks to see what object it is and calls the appropriate method, which is equivalent to a virtual function in C++. This involves overhead: the program must first examine the object to determine its type, select the appropriate method, and then call it. Calling a method is quite slow compared to executing instructions within a method. In fact, as my benchmarks show, a loop that executes n times doing nothing but counting is 50 times faster than one that calls a method that does nothing. (See the benchmark on my Web site, www.righttrak.com/javaperformance/benchmarks.)

What is the cost of a method call and how can you reduce it? A static method costs about three units of time on my Pentium 4 PC, where a unit is defined by an empty counting loop. A final method costs roughly the same, and an ordinary nonfinal method is about three times as expensive. The conclusion is obvious: whenever possible, use the final or static qualifiers on methods (in other words, if you don't intend to override the method, say so).

Let's start by saying that if you want a program to run fast, get JDK 1.4 and run it with optimization turned on:

```
java -server MyClass
```

The `-server` option scans the entire loaded program as it's being run, eliminating methods by inlining them, turning methods into native assemblers, removing constant evaluations from loops, and other optimizations. It improves performance, often by a factor of 10 in CPU-intensive bits of code. It might surprise you to think about optimizing programs at runtime, but considering that Java runs on different machines, the only way to optimize for your particular processor is at runtime.

This feature is new in 1.4. There's a "bad feature" in 1.3 that tends to invoke the JIT compiler lazily, often too late. If you

compile the following program and run it under 1.3, any code in main is optimized, but code in f() is optimized only after it's called once:

```
public static void f() { ... }
public static void main(String args[]) {
    f();
}
```

In this case, since f() is called only once, it's obvious that the compiler had better optimize f before executing it, or not bother.

The compiler makes certain assumptions about what is worth inlining based on the fact that inlining code can take more memory if the code is big. If you have a big routine, it won't be inlined. This can be very unfortunate in some specialized cases, which is where human intelligence comes in.

Suppose you're writing f(), which calls g(), which calls h(). You're doing this just to break up the code and make it easier to read. However, even though f is the only function that calls g(), if g is big enough it won't be inlined. It doesn't matter if there's a big loop involved:

```
void f() { g(); }

void g() {
    for (int i = 0; i < 100000000; i++)
        ...;
}
```

because the time it takes to perform the loop dwarfs the time it takes to get in and out, and so the percentage cost of the procedure call is tiny.

There is, however, a case to be made for applying human intelligence to inlining code. Frequently a method is large,

```
return sum;
}
```

In the second case, f8 calls the logging routine only if debug mode is on. At the cost of an extra statement every time you call the log routine, this code runs 25% faster.

```
public static int f8(int n) {
    int sum = 0;
    for (int i = 0; i < n; i++) {
        if (debug)
            log2(i);
    }
    return sum;
}
```

Multithreading and Synchronization

Since Java is a multithreaded language, the synchronized primitive is provided to make sure that multiple threads of execution do not destroy objects. When entering a synchronized method, it acquires a lock that's associated with the object and prevents any other synchronized method from entering. Acquiring such a lock is a slow machine language instruction. The result is that calling a synchronized method is three times as slow as an ordinary method, which in turn is three times as slow as a static or final method. The computer must first check whether someone else already has the lock, and if not, acquire the lock all in one atomic operation.

```
public synchronized void f() { ... }
```

Multithreading is a complex topic, and the reader is well advised to read one of the many books on the topic for a full understanding. However, a very quick overview of optimization should begin with the observation that since acquiring locks and managing them is a costly business, we should



“

Since Java is a multithreaded language, the synchronized primitive is provided to make sure that multiple threads of execution do not destroy objects”

but the first line of the method is a test that determines whether or not to execute the rest. Consider a logging routine. If debug mode is on, it should write out. But for all those cases where debug is false, why call the routine at all? Here's an example of this in action:

```
private static boolean debug;
```

In the first case, f7 calls the logging routine regardless of the state of the debugging flag:

```
public static int f7(int n) {
    int sum = 0;
    for (int i = 0; i < n; i++) {
        log(i);
    }
}
```

avoid it unless there is some real reason for their use. Furthermore, multithreading will only result in a gain of efficiency if we can overcome the immediate loss of efficiency that results from calling synchronized methods. In a great many situations, multithreading is not called for, and the simplest and best way to handle the situation is to say that the object is not thread-safe, and that programmers should never make two simultaneous calls to the same object.

On the other hand, if multithreading has a significant advantage, the best way to achieve it, where possible, is to have more than one object and give each thread its own object. As perfect examples of this, consider IO streams in a Web environment. While there may be many threads simultaneously writing Web pages, each one is writing to its own Web page. In such cases, the IO stream for servlets could be implemented as a fast, unsynchronized version of PrintStream.

engenuity
engenuity.com



Sometimes, however, there are applications (like a log) where it's vital that multiple threads be able to write to the same object. In such cases, synchronization is vital for correctness. While we can add a new class to the library to support unthread-safe IO, we must always continue to support thread-safe IO for those few cases where it's important.

If you're going to acquire a lock, do so only once. Planning how locks are acquired and released is not only good optimization practice, it's worth really thinking over as this is one of those tricky areas where badly thought-out designs are not only slow, but often don't work in very subtle, nonrepeatable ways. These are the hardest possible situations to debug. Because acquiring the lock means that no one else can enter, synchronized critical sections should:

1. Be as short as possible
2. Not call other synchronized routines (i.e., do whatever needs to be done in a single synchronized section if possible)
3. Never allow unsynchronized access to critical data
4. Never deadlock

Case Study

Simply removing all the synchronization from `java.io.PrintWriter` and writing a class that is functionally equivalent but not thread safe resulted in a 50% improvement in speed. Class `PrintWriter` contains synchronized methods that call other synchronized methods, in some cases three deep. The long chains of method invocation before getting to any actual code is a large part of what slows down IO.

Calling Native Methods

You might assume that if you really need speed, you can resort to linking in some C++ code and call that for the ultimate in performance. The answer may surprise you; it certainly surprised me. Even ignoring the obvious disadvantages of using C++ – the lack of portability, requiring a shared library to deploy an application, etc. – the simple fact is that calling a native method is twice as slow as an ordinary method call.

Having looked a bit at the implementation of the JDK, I can tell you that while it may be tweaked a bit, the reason is essentially sound – to call a C++ routine, you must first make a native mode call (that's one) and then set up a call to the underlying C++ routine; twice as much work, twice as much time, right? And to communicate with anything in the Java environment takes further calls as well, so the only way you'll see a significant speed advantage is by staying in the C++ world for a while. In short, native methods seem to be totally outclassed at this point by a combination of increasingly good optimization in the Java world and the somewhat inefficient code involved in the communication between the two.

Creating Objects

As a C++ programmer originally, I assumed that the biggest cost I was likely to find was the synchronized method call. I was surprised – the slowest operation by far was the creation of an object. In hindsight it makes perfect sense. Creating an object requires the allocation of memory, including all the overhead for identifying the class of the object, its lock, and the amount of memory being used. After using the object for a time and invoking methods, the garbage collector must eventually free the memory that has been allocated. The act of allocating the memory alone, even when optimized in JDK 1.4, is far more expensive than a synchronized method call. The overriding rule in Java code optimization is simple: don't create unnecessary temporary objects.

In the following example, the first version, which creates

only a single object and repeatedly queries it, is 800ms versus 26,300ms, or more than 30 times faster than the second one, which repeatedly re-creates the object. This is an extreme example, of course, because what is being done is very simple compared to the object creation, but it gives an idea of just how costly object creation is.

```
public static int f10(int n) {
    int sum = 0;
    TempThing t = new TempThing(0);
    for (int i = 0; i < n; i++)
        sum += t.getV();
    return sum;
}

public static int f11(int n) {
    int sum = 0;
    for (int i = 0; i < n; i++) {
        TempThing t = new TempThing(0);
        sum += t.getV();
    }
    return sum;
}
```

Case Study

While removing synchronization and streamlining the code path of `PrintWriter` resulted in a factor of two improvements in performance, eliminating the temporary string created in printing an int resulted in a sixfold performance improvement.

String Manipulation

Many programmers have seen the sequence:

```
String s = "a" + "b" + "c";
```

and know that `StringBuffer` is better:

```
StringBuffer b = new StringBuffer();
b.append("a").append("b").append("c");
```

This knowledge seems to break down after this point. If you're processing large strings in `StringBuffers`, don't then turn them back into strings to pass them to another routine unless you're worried about multithreading problems. As long as you're processing single threaded, you're better off continuing to append into the `StringBuffer` until you're done. The following routine:

```
public String getAsXML() {
    StringBuffer b = new StringBuffer();
    b.append(...);
    return b.toString();
}
```

must make an unnecessary copy in order to turn the `StringBuffer` into a string. Then, if the caller is going to append more text, this string must be appended into yet another `StringBuffer`. This is a big waste. Instead, try:

```
public void getAsXML(StringBuffer buf) {
    buf.append(...);
}
```

where the caller allocates the `StringBuffer` and passes it to the routine, which fills it. The caller can then continue processing.

nsoftware
nsoftware.com

This approach has another advantage, namely that the caller usually has a much better idea of the total size of the StringBuffer at the end of processing. It is vastly more efficient, if you know how many characters are involved, to preallocate them rather than allow the StringBuffer to start at the default size of 16 and grow, which requires a lot of copying. For example, if you know the eventual size of the string will be as high as 2K, then:

```
StringBuffer buf = new StringBuffer(2048);
obj.getAsXML(buf);
```

will typically result in approximately 100% performance improvement over the original string code. It's far better to overallocate than to underallocate and require a grow operation. Remember, this works only if the string in question is not being assaulted by multiple threads.

Manipulating strings, even optimized ones, takes a fair amount of work and code, even if the string length is one. If you're processing a single character, using a char is much faster, so:

```
buf.append('\n');
```

is significantly faster than:

```
buf.append("\n");
```

Efficient Use of Lists

Java provides a fairly rich set of data structures. They're not all the same, and while they may work interchangeably, that doesn't mean they're all equally good in all circumstances. To build up a list in order, ArrayList is faster than LinkedList by a factor of two. LinkedList is substantially slower because each

later, even if you overallocate.

Last, remembering that object allocation is the slowest activity of all, you can easily see that this list, which must create object wrappers for each int, is vastly inefficient. The following code, using a list class written just for int elements (see my Web site for the code), runs a full four-and-a-half times faster than ArrayList.

```
IntArrayList a = new IntArrayList(n);
for (int i = 0; i < n; i++)
    a.add(i);
```

For scanning through an existing list, ArrayList is the fastest of the JDK list classes; getting an element from an array is a trivial operation, so synchronization dominates the time. Here, LinkedList can be monstrously slow if you use it incorrectly. Since LinkedList is not a random-access data structure, calling get(i) means it must start from element 0 and scan forward until it reaches position i. A loop that scans through the entire list is therefore not an O(n) operation, but O(n²). For a list of 100,000 elements, my computer performed the ArrayList traversal in 3.25 milliseconds. LinkedList traversal took an astounding 113,657 milliseconds, or 34,971 times slower.

```
LinkedList l = ll;
for (int i = 0; i < n; i++)
    l.get(i);
```

The correct way to code traversal through a LinkedList is to use the iterator design pattern:

```
LinkedList l = ll;
for (Iterator i = l.iterator(); i.hasNext(); )
    i.next();
```



Just choosing the right data structure for your situation can pay enormous dividends"

node requires the creation of an object. Vector is a close second in speed; it's slower because it's a synchronized data structure. However, in situations where values are to be inserted in the middle of the list (or worse still, the beginning), LinkedList is the best by orders of magnitude since it does not have to constantly copy elements to move them aside.

```
ArrayList v = new ArrayList(n);
for (int i = 0; i < n; i++)
    v.add(new Integer(i));
return v.size();
```

While both Vector and ArrayList use a doubling algorithm that will adaptively grab larger and larger chunks every time the size is exceeded, each time they grow an enormous expense is incurred. As with StringBuffer, it's about twice as fast to preallocate as much space as you'll need than to grow

The lesson to be learned here is that it pays to understand your data structures well. Just choosing the right data structure for your situation can pay enormous dividends. And using one incorrectly, as in the case of LinkedList traversal, can be very costly.

Last, if you want to store a list of primitives, the best way would be to have classes designed for the purpose, like IntArrayList. No one wants to go to the expense of writing and maintaining all permutations of lists for all the primitive data types; this is one reason Java needs a high-quality template facility like C++. That's a topic for another day, but one that I hope to revisit in a future article. For now, a friend and I are proposing some primitive list classes to add to the Java library, because when you do want a list of primitives, there's no substitute for a decent data structure.

Maps

HashMap is quite a bit faster than the older Hashtable, mostly by virtue of not being synchronized. However, the

actuate
www.actuate.com

algorithm used is still less than optimal. To analyze it further, you have to look into HashMap's source code, and know a bit about hashing algorithms. In general, a hash algorithm is fast because it "hashes" the key and turns it directly into the location of the bin where the value is stored, making it an O(1) operation. The problem comes when two different keys happen to hash to the same bin. Statistically, this happens fairly often, and it's the job of the writer of the HashMap to reduce it as much as possible.

Collisions cannot be totally eliminated in the general case, so the design of hash algorithms must allow for them. Therefore, each bin in the HashMap is essentially a linked list for all the keys and values that could hypothetically end up there. This means that every time you add an element to a HashMap, you're once again creating an object that holds the key, the value, and a reference to the next node in case any more values happen to land in the same bin.

Object creation is the most expensive operation possible, so I've tried a different approach and have on my site a couple of experimental classes that perform twice as fast as HashMap (FastHashMap) or four times as fast if your key is an int (FastIntHashMap). They do, however, achieve part of their spectacular speed by not checking the size each time a new element is added, so you must allocate the right size table in advance.

As with all other Java data structures, if you add too many elements to a Hashtable or HashMap, they grow. This is the worst thing you can do, since growing requires painfully reinserting every element. Hashing requires about 25-30% more bins than there are elements for efficient operation. Always preallocate what you think is the right size for your Hashtable, be generous, and check at the end to be sure you were right and that the table did not have to grow.

Last, because the hash algorithm for strings looks at every character in the string, avoid hashing large strings if at all possible. The smaller the string, the faster the hash.

Strength Reduction

Turning slow machine language instructions into equivalent but faster ones is traditionally the job of a peephole optimizer in a compiler; the optimizer looks at a window of instructions coming out of the code generator and makes judicious substitutions. In the Java environment there are two stages at which peephole optimizations can be done. One is during compile time when the source code is turned into JVM code; the other is when the code is run and the JIT turns JVM code into a native assembler. The latter is the approach chosen by Sun, because that way they can optimize code for the particular processor running the code.

Having admitted that most strength reductions are things compilers should do, if your compiler doesn't do them (and Java didn't used to), then it's up to you to do them yourself. In doing so, there are a number of issues: Will the resulting code be as simple as or simpler than the original? Gaining a little speed while losing understandability is not a great bargain. Will the resulting code be faster? Programmers often assume they're optimizing, when in fact they're doing the reverse. The kind of clock cycle counting is certainly better done by a compiler, with knowledge of the target CPU and environment if at all possible. The good news is JDK 1.4 now does some strength reduction. It's up to you to decide how much speed you need now.

First, what not to do. Multiplications by the constant power of 2 are automatically converted to shifts by the computer:

```
x * 2      x << 1
x * 16     x << 4
```

More complex, but not worth it, are multiplications by constants:

```
x * 10      (x << 3) + (x << 1)
```

Divisions are not supported at the moment, but will be soon. If you need the speed right now, the speed of the division itself is five or six times faster.

```
x / 2      x >> 1
```

A much more important strength reduction, and one that the JIT is not likely to detect in the near future, also involves division. Often, programmers want to go around a loop, but do something different every *n* times. One standard trick is to count and take the counter modulo *n*, as in the following example:

```
for (int i = 0; i < 100000; i++)
    if (i%10 == 9) {
        // do something every tenth time
    }
```

This is slow; the following is four to five times faster:

```
for (int i = 0, j = 10; i < 100000; i++,j--) {
    if (j == 0) {
        // do something every tenth time
        j = 10; // restart the count
    }
```

Similarly, if you have code in a loop like:

```
j = (j + 1) % n; // j should always end up between 0 and n-1
```

it's much faster to write:

```
if (++j == n)
    j = 0;
```

In general, for any positive number *x*, $x \% n$ is equivalent to $x \& (n-1)$ if *n* is a power of two. So $x \% 8 == x \& 7$ as long as *x* is positive. Using the & operator is a lot faster.

Summary

All the performance enhancements in this article have involved the application of simple techniques to make individual sections of code faster. If you learn these tricks and apply them everywhere as a matter of course, your code can get significantly faster without a lot of effort. These techniques, combined with JDK 1.4 and the next generation of Java compilers, are going to take us within a hair's breadth of being as fast as a well-written C++ application – and most applications in C++ are not well written. The world will enjoy the resulting crisp handling of the programs to come. Get out there and write something great. ☺

AUTHOR BIO

Dov Kruger is president of Right TRAK, Inc., a consulting and training company focusing on Java, object-oriented, and Web-based technologies. He's currently working on improving the performance of dynamic Web pages with graphics, internationalization, and some of the Java libraries.

fiorano
www.fiorano.com



JASON R. BRIGGS J2ME EDITOR

The Computer of Tomorrow

At times, I wonder just how far short the computer industry has fallen of its promise of a few decades ago. I'm not talking about the lofty ideal of the computer of the future that science fiction authors were predicting we'd be using by now, such as machines capable of holding a proper conversation (or better yet, capable of withering sarcasm in the face of human stupidity), human-computer symbiosis, etc. Nor am I talking about retro chic – the “style of the future” that artists were drawing back in the '50s – and that Apple seems to have captured so well in a lot of their products.

No, it's interoperability that's on my mind.

My wife and I recently bought a house, and while we haven't moved in yet, she's already planning the color scheme for every room, as well as 10 years of renovations. I, on the other hand, have been thinking about networking.

We both have laptops. I also have a Linux server (currently sitting at my parent's house doing nothing), plus various other bits and pieces of hardware (a printer, etc.).

Here's where wishful thinking about those “computers of the future” comes in. I'd like to be able to plug any device into a port in any room and have it all work instantly without any messy configuration or setup.

I know what you're thinking, and it's probably along the lines of Jini, Bluetooth, 802.11, Home Powerline Networking, etc., etc. Certainly with one (or a combination of) those technologies, I can approach the idea of what I want. It certainly won't be a case of plugging it in and forgetting about it, and that's a shame.

I can see some of the reasons behind the lack, of course. Home Networking hasn't exactly had worldwide penetration compared to the Internet, for example. Only a small percentage of the population has wired

everything and the doghouse together (or set up a wireless backbone, for that matter).

The expense is another good reason and I'll give you an example: a couple of years ago I purchased a USB scanner from a well-known computer and peripherals conglomerate. At the time I was primarily running Windows 98 SE and had no problem installing the device. Go forward one iteration of Windows (to the nightmarish Millennium Edition), and there seems to be no way to get that scanner to work anymore. That's just one version of Windows, so I'm guessing my chances of it running on Linux are somewhere between nil and nada. As for “just plugging it straight into the network...,” forget about it.

Obviously, in this case the manufacturer has offloaded a lot of the scanner functionality onto the computer (in the form of one gung ho 38M driver installation, I might add) to save costs – so really, it's my own fault for buying cheap. What if the scanner had its own small processor, storage, network card, and, possibly, Web server (to access the pictures)? While it suddenly becomes a network device I can plug anywhere in my home network, it also becomes a darn sight more expensive. Then again, with something like Dallas Semiconductor's TINI board (Java-enabled) which, I seem to recall hearing, costs around U.S.\$50, it doesn't seem that much more expensive to add some of those features (see www.ibutton.com/TINI/index.html for more info).

Just think of the advantages of having all these network-ready (and why not Java-enabled) devices. How about a (waterproof?) PDA in the kitchen with a Bluetooth connection to the (wired) network port for looking up recipes. Or intelligent speakers – plugged into any network port, they'll have access to the house's music repository. Taking it one step further, I want to be able to plug a digital camera into any port in my house and have the

The Computer of Tomorrow

How far short has the computer industry fallen of its promise of a few decades ago? I'm not talking about the computer of the future that sci-fi authors were predicting we'd be using by now, nor about retro chic. No, it's interoperability that's on my mind.

by Jason R. Briggs

54

Does J2ME Have Its Legs Yet?

Sun has poured a lot of resources into the J2ME platform, recognizing that the next battleground will be the ubiquitous consumer device. Whether J2ME can make the huge impact that Sun hopes for is still an open question.

by Matthew Ferris

56

Wireless J2ME Applications with Java and Bluetooth

This article shows how to write wireless J2ME applications using Bluetooth. If you've heard of Bluetooth, you've certainly heard of 802.11b (the wireless LAN protocol) – both of which are wireless communication protocols.

by Bruce Hopkins

60

photos automatically upload onto a specially designated shared resource ready for access. Or how about a DVD recorder that asks any connected devices what critical information they need backed up, then automatically does it and sends out a “distress” signal (e-mail, phone call, SMS message) when it needs a replacement disk.

These are the kinds of things that actually make computers more convenient, less of a pain for the ordinary person to use – and come close to that promise of yesteryear. ☛

▼▼▼ jasonbriggs@sys-con.com

AUTHOR BIO

Jason R. Briggs is a Java analyst programmer and – sometimes – architect. He's been officially developing in Java for almost four years, “unofficially for five.”

inetsoft
www.inetsoft.com

Does J2ME Have Its Legs Yet?

How to take advantage of mobile device programming



WRITTEN BY
MATTHEW FERRIS

Sun has poured a lot of resources into the Java 2 Micro Edition platform, recognizing that the next battleground will be the ubiquitous consumer device. Whether J2ME can make the huge impact that Sun (and the developer community) hopes for is still an open question, as the current rate of adoption has been underwhelming thus far.

All the major OEMs have announced big plans for J2ME, and although the sluggish economy of late can be factored in as one reason for J2ME's slow growth, there are several other contributing factors. I'd like to offer some suggestions about what's wrong and how to fix it.

1. *Supply and demand disconnect*

It's been noted that although most of the major financial firms have rolled out wireless products to allow their customers to trade using their PDAs or other consumer devices, demand from customers has been almost nonexistent. Vendors' attempts to drive demand is not new – major software manufacturers have been doing it for years. Many people probably have enough AOL disks to tile their bathroom.

In the case of financial institutions, it's particularly ironic. For years the sage advice has been to get in the market and stay in, don't try to time it. Yet if I have to execute a trade in the cab, it's a sure bet I'm a market timer.

New technologies typically have a first-generation incarnation centered on games, as is the case with J2ME. Yet, more than one company that focused on this market has already gone belly-up. Once it's demonstrated that the functionality for more compelling applications exists, the games become somewhat of a yawn. While the PS2 and Nintendo games certainly have mileage

left in them, there are two important differences. These devices are built specifically for gaming, and while they're adding other functionality, the core functionality remains just that. In addition, it will be a long time before portable consumer devices have anywhere near the fire power to present graphics or redraw rates that cause similar jaw-dropping reactions. (Nor have I heard of any OEMs planning a phone with a 27-inch screen anytime soon.) I'm not saying that you can't make money on games, simply that a mature market for J2ME will mean far more than just games.

It's time that vendors who are focused on J2ME get beyond games and on to those applications that provide unique value precisely because they are for the phone.

2. *Carrier reticence*

Even though carriers and vendors may know what consumers want and have the ability to provide it, doing it is quite another matter. For years, carriers have thought that their network was everything. They've spent lots of capital building these networks, and anything or anyone that could possibly cause a disruption was not an ally but a threat – so developers are viewed in the eyes of some carriers. They don't want applications sucking up bandwidth from paying (phone) customers.

Nextel was the first carrier in North America to support J2ME, yet they haven't exactly smoothed the road for developers to get their apps out there in the hands of the public. To be fair, there are concerns that need to be addressed. An application that causes grief for a phone customer may end up costing the carrier valuable support resources. However, the way around this has been to subject developers to high entry barriers in order to get their applications out. Nextel has a hefty certification fee that you must pay to have an application checked for any rogue code (although assessing the fee is at the discretion of Nextel). However, J2ME's security model is such that it's very difficult to do anything that would impair the hardware in any major way. Indeed, functionality outside of J2ME is not accessible to a J2ME application. By requiring some basic compliance, such as timeout values for connections, misuse of the network could be bypassed quite easily.

The other mindset is that end users will want to buy applications for their devices. Carriers need to realize that selling applications as software is a dead-end business. They've been providing voice service for years; why can't they use the same model for applications and treat them as services? The example of NTT DoCoMo in Japan is quite compelling – their service is

sitraka
www.sitraka.com

“What distinguishes location-based services for the phone is that it’s unique to the device”

structured just that way. If a user wants to access an application, he or she pays a few yen per minute for it. The key difference is that they treat this as a service, rather than seeing themselves as software vendors. NTT regards the network as a conduit for the service, not the service itself. As a result, their pattern of thinking is completely different from that of carriers in North America.

Competition is good, and it will be quite interesting to see what happens when Sprint enters the fray later this year with their J2ME support. We can hope that it will be a boon for the devel-

oper community as well as for more widespread use of J2ME.

3. *Location-based services*

It’s been said that the killer app for phones is the Short Message Service (SMS). This may be the case at present, but even SMS is not unique to phones. Using Instant Messaging on the desktop is pretty much the same thing, albeit from a fixed location. What distinguishes location-based services for the phone is that it’s unique to the device. If I’m at a certain corner of the city and want to know where the nearest post office is, or where the nearest outlet to purchase

concert tickets may be, it’s a huge convenience if my phone can determine where I am and direct me to my goal.

These services do exist, but there’s simply no API available for the J2ME developer to get at them. The phone knows where it is all the time, at least within a certain triangle of cell towers. By providing this information to the programmer, the carriers would be doing a huge service to their customer base. Location-based services could be the thing that persuades consumers to buy a particular phone model, or to sign on with a given carrier simply because they have the most compelling services.

There are significant challenges to the widespread adoption of J2ME. Yet Java’s position as not simply a programming language makes it uniquely positioned to take advantage of mobile device programming. A phone or PDA running Java becomes not only a convenient piece of hardware, but a node in a network that takes full advantage of the entire Java environment. What is needed is closer cooperation between the carriers and the developer community in order to bring applications to market that will be a win for both sides. ☛

mferris@objectsinc.com

app dev
www.appdev.com

AUTHOR BIO

Matthew Ferris is the mid-west editor of Wireless Business & Technology and the president of the Chicago Wireless Developer User Group.



J2ME



J2SE



J2EE



Home

zerog
www.zerog.com

written by Bruce Hopkins



This is Part 1 of a two-part article that will show you how to write wireless J2ME applications using Bluetooth. If you're a J2ME developer, this will quite likely be your first introduction to Bluetooth. Perhaps you've heard a lot about it, but you're not sure what it is or how it works. Well, you've come to the right place.

Part 1 of 2

Wireless J2ME Applications with Java and

Bluetooth

First, we'll look at the Bluetooth protocol, then compare it to another common wireless technology: 802.11b. Next, we'll look at the anatomy of a Java Bluetooth-enabled device and examine its components in detail. Later on, we'll dive into the details and explain the purpose of the Bluetooth stack and profiles (not to be confused with J2ME profiles). Finally, we'll wrap up by providing some example code that shows how to initialize your Bluetooth stack according to the Java APIs for Bluetooth (JSR-821.0a).

What Is Bluetooth?

Simply stated, Bluetooth is a wireless communication protocol that you'd use to connect two or more Bluetooth-capable devices. In this sense it's like any other communication protocol, such as HTTP, FTP, SMTP, or IMAP. Bluetooth is also similar to these protocols in that it has a client/server architecture. In Bluetooth, the one who initiates the connection is

master (the client), and the one who receives the connection (the server) is the slave. However, what makes Bluetooth so special is that it's wireless. You can connect two Bluetooth devices to share data or transfer files without using messy cables.

Bluetooth vs 802.11b

If you've heard of Bluetooth, you've certainly heard of 802.11b (the wireless LAN protocol) – both of which are wireless communication protocols. Bluetooth and 802.11b are geared to accomplish two different goals, although the technologies operate in the same frequency band: 2.4GHz. The question many people ask is: If both technologies operate at the same frequency, won't they interfere when placed in range with each other? Not so, according to a Forrester Research study in 2001.

Another common question: If they both do the same thing, won't one eventually replace the other? The key point is that

altova
www.altova.com

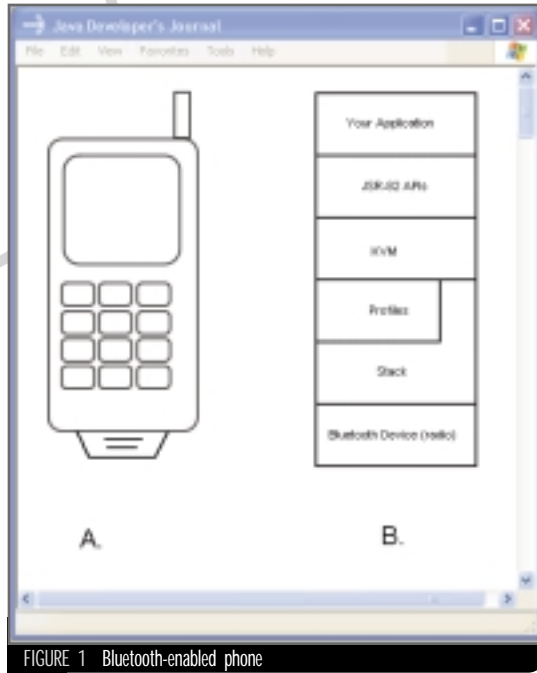


FIGURE 1 Bluetooth-enabled phone

they're not intended for the same purpose. Bluetooth will replace 802.11b (or vice versa) when, for example, the microwave replaces the conventional oven.

Microwave vs Oven

The oven is great at cooking, but bad at heating food quickly – and it's also pretty expensive. Conversely, the microwave is cheap and great at heating food quickly, but it's bad at cooking. Both devices have their trade-offs, although either could be used for heating and cooking. How does all this compare to wireless communication?

It's pretty simple. Wireless LAN (802.11b) is good at connecting two relatively large devices with lots of power at high speeds.

A good use of the technology is connecting two laptops at 11Mb/s. Wireless LAN is also good at connecting those devices at long distances (up to 300 ft).

you're transferring a 50K file between two PDAs?

One of Bluetooth's strengths is its ability to function as a cable replacement technology. If you have multiple peripherals connected to your computer using RS-232 or USB, then Bluetooth is the ideal solution if you want to use those devices wirelessly. It's difficult (if not impossible) to connect computer peripherals using 802.11b technology (except for printers). Bluetooth even has a built-in capability for wireless audio communication. To put things succinctly: Bluetooth will never replace 802.11b because it's bad for:

- Large file transfers between devices
- Long-range communication

On the other hand, 802.11b will never replace Bluetooth because:

- It can't be used to communicate to peripherals.
- It requires too much power for small devices.
- It's overkill for small data transfers.
- It wasn't designed for voice communication.

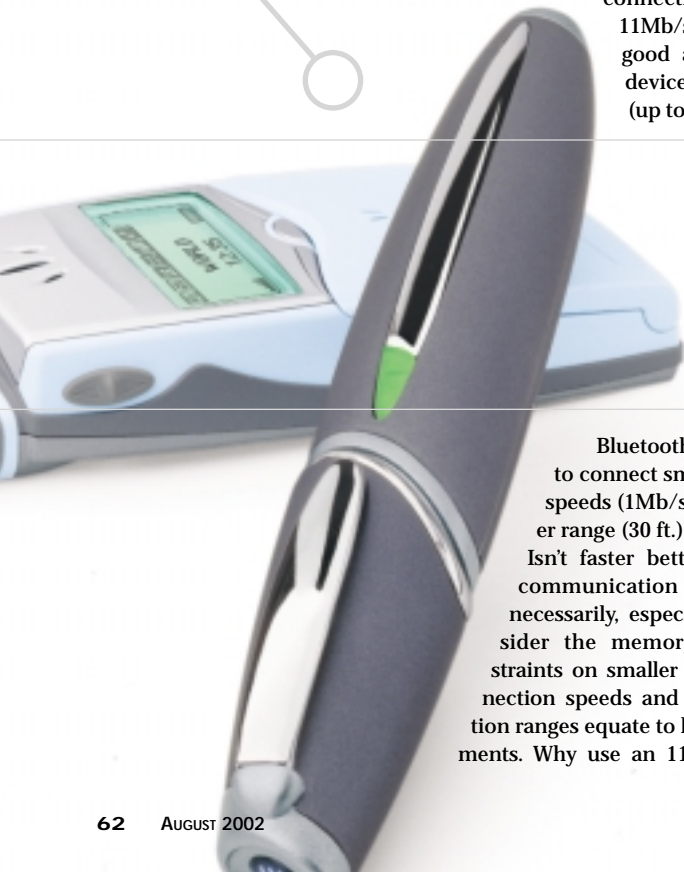
For the moment in the wireless communications arena, there's no technology that's best suited for every possible application. Either Bluetooth or 802.11b can be utilized for wireless communication between computers. Both have their place in the market, and both can perform remarkably well in their niches.

Bluetooth Specification

For Bluetooth devices to communicate properly, they need to conform to the Bluetooth Specification. This specification, like any other spec, defines the standard that a Bluetooth device should adhere to, as well as rules that need to be enforced when communicating. You can download the specification documents from www.bluetooth.com.

Java Bluetooth Application Anatomy

Let's look at the anatomy of a Java Bluetooth application. Figure 1, section A, shows a Bluetooth-enabled phone that includes the CLDC MIDP platform; section B contains the anatomy of the device shown in section A. The bottom layer of the diagram in section B represents the actual Bluetooth device that sends out your voice and data over the air: the Bluetooth radio. Section A shows the Bluetooth device as a peripheral, but it could easily be integrated into the device by



"One of
Bluetooth's
strengths is its ability to function as a cable replacement technology"

Bluetooth is ideal if you want to connect small devices at slower speeds (1Mb/s) and within a shorter range (30 ft.). Why slower speeds? Isn't faster better? Isn't long-range communication a good thing? Not necessarily, especially when you consider the memory and power constraints on smaller devices. Faster connection speeds and longer communication ranges equate to larger power requirements. Why use an 11Mb/s connection if

the OEM. Bluetooth radios come in many shapes and sizes; Figure 2 shows the popular Bluetooth radio made by 3Com. 3Com also makes a handy network browser with their Bluetooth device (see Figure 3).

The Bluetooth Protocol Stack

The next essential component of a Java Bluetooth application is the Bluetooth stack, (again, see Figure 1, section B). The Bluetooth stack is the software layer with direct access to the Bluetooth radio as well as control over such things as device settings, communication parameters, and power levels. The stack consists of many layers, and each layer has a specific task

cita
www.cita.com



FIGURE 2 Bluetooth radio

in the overall Bluetooth device (see Figure 4). The Bluetooth device manufacturer is not required to use all the layers, so let's cover the main ones that should be in most Bluetooth devices.

- **Host Controller Interface (HCI):** The interface between the radio and the host computer
- **Logical Link Control and Adaptation Protocol (L2CAP):** The multiplexer of all data passing through the unit; audio signals, however, have direct access to the HCI
- **Service Discovery Protocol (SDP):** Used to find services on remote Bluetooth devices

"If you want your Bluetooth-enabled devices to interact, having a Bluetooth stack is not good enough: they need to conform to a particular profile"

- **RFCOMM:** Widely known as the virtual serial port protocol
- **OBEX:** Object exchange protocol

Bluetooth Profiles

The next level in Figure 1, section B, is Bluetooth profiles. They were created to allow different Bluetooth devices to interoperate. For instance, let's say you own a Bluetooth-enabled PDA and a Bluetooth-enabled wireless phone. Both devices have Bluetooth stacks. How can you tell if those two devices will allow you to synchronize the phone lists between each one? How will you know if you can send a phone number from the PDA to the phone? And, most important, how can you determine if these devices will allow you to browse the Internet on the PDA using the phone as a wireless modem?

A Bluetooth profile is a designed set of functionality for Bluetooth devices. For instance, using the examples listed above, the phone and the PDA must both support the Synchronization Profile in order to synchronize data between themselves. To send object data like a .vcf from the PDA to the phone, both devices need to have the Object Push Profile implemented. Finally, the PDA and the wireless phone must both support the Dialup Networking Profile in order for the PDA to wirelessly browse the Internet from the phone. If you want your Bluetooth-enabled devices to interact, having a Bluetooth stack is not good enough: they need to conform to a particular profile.

Bluetooth Profiles vs J2ME Profiles

Do not confuse Bluetooth profiles with J2ME profiles. J2ME profiles are a set of Java classes that extend the functionality of a J2ME configuration. For example, both the PDA and MID profiles are a set of Java classes that extend the functionality of the Connected Limited Device Configuration.

A Bluetooth profile can be implemented in any language and on any platform because it refers to a defined set of functionality for a Bluetooth-enabled device. So the Object Push Profile can be implemented on a Palm OS PDA in C++ as well as on a Bluetooth-enabled printer in Assembler;

qualcomm
qualcomm.com

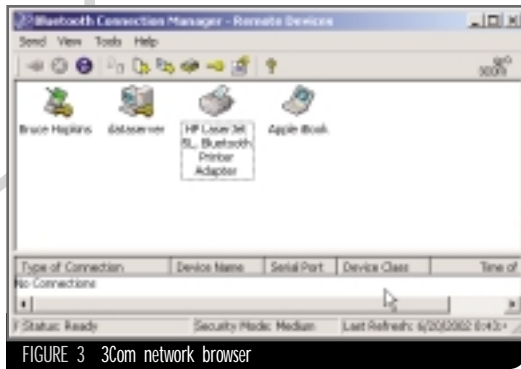


FIGURE 3 3Com network browser

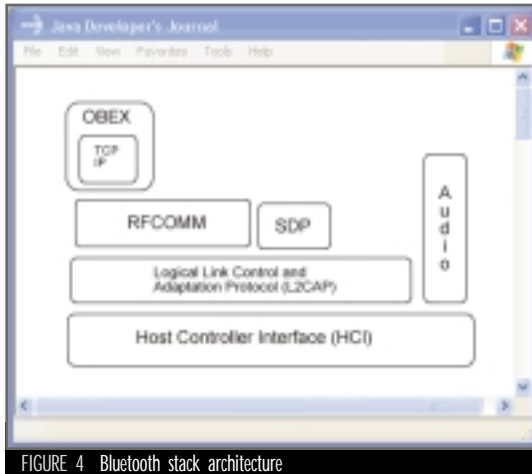


FIGURE 4 Bluetooth stack architecture

“Stack initialization can consist of a number of things, but its main purpose is to get the

Bluetooth

device ready to start wireless communication”

it's just a defined set of functionality.

The next two layers of software in Figure 1, section B, are the KVM and the Java Bluetooth API libraries. With J2ME and the standard Java Bluetooth implementation, you have direct access to the profiles and the stack of your Bluetooth device. The Java Bluetooth API (JSR-82) works with any Java platform that supports the Generic Connection Framework (GCF). Unfortunately, that means you can't use J2SE with the official Java Bluetooth API. Hopefully, the GCF will be incorporated into JDK 1.5. Until then, you'll have to use nonstandard libraries if you want to write Java Bluetooth code for desktop applications. The last layer in Figure 1, section B, is your application code, the wireless killer app that will make you famous.

Java Bluetooth Application Concepts

The basic concepts of any Bluetooth application (Java or otherwise) consist of the following components:

- Stack initialization
- Device management
- Device discovery
- Service discovery
- Service registration
- Communication

The Java Bluetooth Specification adds a special component to the mix called the Bluetooth Control Center (BCC), which is outside the scope of this article.

Stack Initialization

Before you can do anything, you need to initialize your stack. Remember, the stack is the piece of software that controls your Bluetooth device. Stack initialization can consist of a number of things, but its main purpose is to get the Bluetooth device ready to start wireless communication. Every vendor handles stack initialization differently; Listing 1 shows how to initialize the stack using the Atinav Java Bluetooth SDK.

Summary

Hopefully, this article has stimulated your interest in application development with Java and Bluetooth. Now that the preliminaries are out of the way, in Part 2 we can dive into more application code that will show you how to wirelessly communicate between two Java Bluetooth-enabled devices. Don't worry if you don't have any Bluetooth equipment yet; I'll also show where and how to obtain the hardware and software you'll need. ☛

AUTHOR BIO

Bruce Hopkins is a senior Java consultant at Great Lakes Technologies Group in Southfield, MI. He has worked with Java for over six years, and has researched in wireless networking for four. Bruce is the coauthor of an upcoming book entitled Java and Bluetooth by Apress (November 2002).

Listing 1

```
import javax.bluetooth.*;
import javax.microedition.io.*;
import com.atinav.bcc.*;

public class WirelessDevice implements DiscoveryListener {

    LocalDevice localDevice = null;

    public WirelessDevice () {
        //setting the port number using Atinav's BCC
        BCC.setPortName("COM1");

        //setting the baud rate using Atinav's BCC
        BCC.setBaudRate(57600);

        //connectable mode using Atinav's BCC
        BCC.setConnectable(true);

        //Set discoverable mode using Atinav's BCC
        BCC.setDiscoverable(DiscoveryAgent.GIAC);

        try {
            localDevice = LocalDevice.getLocalDevice();
        } catch (BluetoothStateException exp) {
        }

        // implementation of methods in DiscoveryListener class
        // of javax.bluetooth goes here

        // now do some work
    }
}
```

Download the Code!
www.javaDevelopersJournal.com

sprint
www.sprint.com

Small Worlds 1.5

by Information Laboratory, Inc.

REVIEWED BY DALE CHURCHETT dale.churchett@salion.com

info

Information Laboratory, Inc.

E-mail: info@thesmallworlds.com

Web: www.thesmallworlds.com

Phone: 917 494-0840

Specifications

Platforms: Java 1.3 and 1.4 applications on Windows 98, NT, 2000, and XP; Linux; Solaris; and any Java-enabled platform

Pricing: Small Worlds Analyzer Edition for Java is \$1,750. Small Worlds Visualizer Edition for Java is \$750

Test Platforms

Computer: Dell Inspiron Laptop 7500
Processors: 400MHz Intel, Red Hat Linux 7.1 Dual 800MHz Intel
Memory: 128MB RAM, 512MB RAM
Platform: Windows 2000

It takes more than a group of keen developers coding like mad to create a software system that meets requirements yet is robust to change. As new requirements are discovered, new code must be written and existing code maintained. Without careful consideration of code structure, packaging, and component dependencies, a large system can quickly turn into a “big ball of mud.”

Managing dependencies between components becomes critical if a software team is to maintain the level of agility required in today’s environment, where customers are demanding increasingly complex business systems delivered faster and cheaper. If component and package dependencies are not well managed, the code base becomes fragile and unmaintainable, a common cause of a software project failure.

Currently, tool support to help developers and architects prevent system degradation is poor. The type of support required is where the feature set of most CASE tools stops. Luckily for us a new breed of tool is now available – Small Worlds.

Product Description

Small Worlds provides insight into the structure of Java or C++ software systems through a variety of innovative visual models and statistics. These include views for managing component and package dependencies, tracking the effects of change, system coherency, and summary reports. The views help identify problem areas of the code that may need system-level refactoring.

There’s also a plug-in API that allows integration into other tools (Forte being the first), and several export options that save visual representations of the system in GIF or HTML format.

Small Worlds supports three modeling notations, or “skins”: UML, Small Worlds, and Global. The UML skin does what you would expect. The Small Worlds skin focuses on the code “flow” rather than the containment hierarchy (i.e., the direction of change rather than associations). The Global skin removes visual decorations to simplify complex diagrams.

The Review

To see how Small Worlds handles a large system, I decided to import the source code from my current project – a J2EE-based application with 230,000 lines of Java code in 1,320 files. On my modest laptop it took three minutes for the compiled code to be imported

and then another two for the views to be created.

Reading through the informative Small Worlds user manual pays off! There are so many features and ways information is presented that you can easily lose sight of what you need to accomplish. There is a big “wow” factor with this tool, which makes it easy to get sidetracked. Therefore, it’s very important that you have a clear understanding of the questions you want answered before you start navigating through your code.

The heart of Small Worlds is the Explorer window that presents a visual representation of system components; convenient navigation between components; and a set of fine-grained filter controls to hide abstractions, implementation, different kinds of dependencies (uses, extends), packages, and classes. Changing to the Small Worlds skin provides a view that illustrates the flow of the code, which is useful for depicting the effects of change.

Apart from the Explorer, the feature I was most eager to look at was the summary report. There I was told that our system was 98% stable, but that there were problem areas (no surprise there), two possibly serious. The first is a Local Hub, a component that has many immediate dependencies and dependents. The component in question was one I suspected was a problem, and now I had evidence. Drilling into the “What If” view immediately demonstrated the rippling effect this component causes when modified. Lowering the threshold of hub dependencies by using the “Show run play” option exposed other hubs I would have to deal with next.

The next problem I investigated was a tangle, a cyclic loop between components. Nothing good ever comes from cyclic dependencies, especially if they’re between classes. Drilling into the Explorer view showed me that the tangle was package-based and needed investigating (see Figure 1).

Viewing the details of the tangle exposed 24 loops that could be broken. Each loop has a weight associated with it that suggests if the dependency is a weak or strong link. With this information and the powerful “What If” feature, I can make educated recommendations to the development team as to how to make this section of the code more robust to change.

Summary

There is no other tool on the market that deals with the complexity of managing component dependencies as completely as Small Worlds. You can reverse-engineer class diagrams in many visual UML tools, and even

purchase some plug-ins for Rational Rose, but none provide a feature set anywhere close to that of Small Worlds.

To take full advantage of the insights provided by the tool and apply them to a large software system, you need an expert user and a strong individual who has the commitment of the entire development organization. However, novice or intermediate-level developers may take advantage of the Visualizer edition that removes the analysis tools and concentrates on navigating through the system structure. For example, the “Random explore all” feature from the Explorer is a great way for new developers to get a feel for the system.

I found this to be an exciting tool and one that I’ve already used to good effect. The ROI is somewhat intangible unless you’re into tracking metrics in your development process, but Small Worlds can help with that too. I do have some minor usability issues with the tool, and the user manual bundled with the installation was not current, but none of this prevents me from highly recommending Small Worlds to others.

References

- Foote, B., and Yoder, J. (1999). “Big Ball Of Mud.” Department of Computer Science, University of Illinois at Urbana-Champaign.
- Churchett, D., and Burhdorf, R. (2002). “The Salion Development Approach: Post Iteration Inspections for Refactoring (PIIR).” *The Rational Edge*. ☪

Small Worlds Snapshot

Target Audience: Architects, component developers, test engineers

Level: Medium to expert

Pros:

- Concentrates on being an analysis tool, not an IDE
- Works well out of the box
- Provides insights not currently available with existing tools in one unified solution
- Great graphical representation offering visual clues to system structure
- Good context-sensitive help

Cons:

- The .avi tutorial did not work
- The Windows batch script to increase heap size did not work
- Potentially steep learning curve/high barrier to entry for Analyzer Edition
- Would need to be installed into the development process to track progress over time
- User manual not in sync with installed version, but good enough

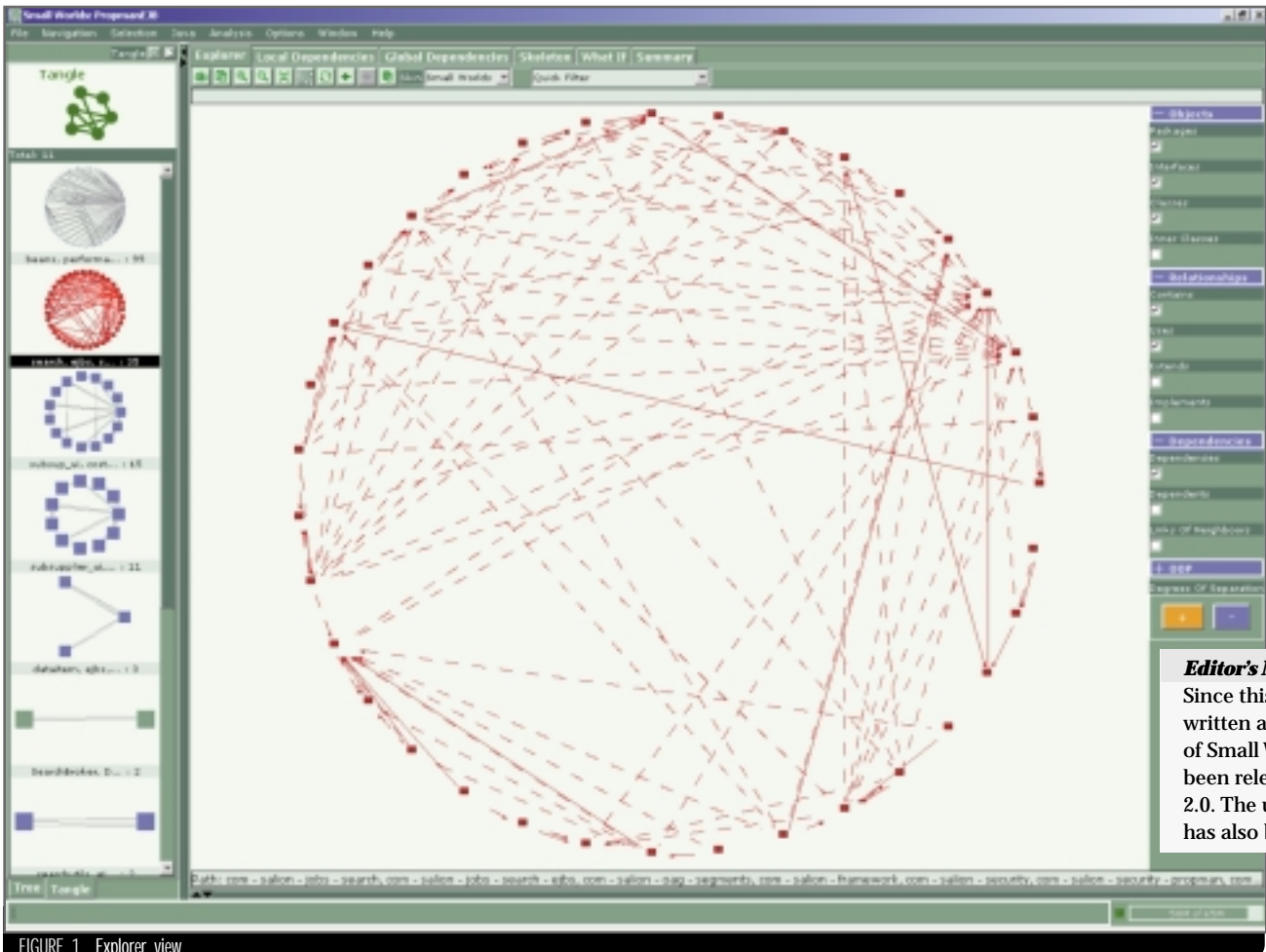


FIGURE 1 Explorer view

Editor’s Note: Since this review was written a new version of Small Worlds has been released, version 2.0. The users manual has also been updated.

Navigation icons: Home, Labs, J2ME, J2SE, J2EE



web services
conference & expo

EDGE

Show Report

WRITTEN BY STEVEN BERKOWITZ

While I wandered, head down, among the detritus of the post dot-com era, a new world order was forming. When I looked up, I found the technical world rallying under a new banner – Web services. Savior or hype? I had to know. What better place to learn than the world's largest Web services event? So off I went to SYS-CON's Web Services Edge 2002 East - International Web Services Conference & Expo. While this was also the JDJEdge and XML Edge Expos, the main thrust was Web services. The keynotes all centered on Web services and most of the Java and XML

Program.

The Monday was dedicated solely to Java University. I took the "Web Component Developer: Certification Fast Path" course about servlet and JSP

cerned about this, but unless day one was very different than the next two, I don't believe I missed too much. While I learned a lot of specific things on the Wednesday and Thursday of the expo, a

"I took the 'Web Component Developer: Certification Fast Path' course...It met its goal admirably, and clarified, for me, a number of issues I was unclear on in these technologies"

technology. Any code-level course where you don't get to write code is at a disadvantage. But, for what it was, I found the course very useful. It's stated goal was to cover the material that's required on the certification exam and give pointers for further study. It met this goal admirably, and clarified, for me, a number of issues I was unclear on in these technologies.

Among the technologies that had passed me by was XML. Sure, I've dabbled in it here and there, and edited my deployment descriptor files, but I never did anything meaty with it.

So Tuesday found me in the day-long "Developing Solutions Using Java Technology and XML" course, presented by Todd Greanier of Compass Technologies. This course was fantastic. It started with a very thorough XML primer, which was exactly what I needed. Then it detailed which Java technologies are available for use with XML. Overall, I was very impressed with this course and its presentation.

The trade-off for taking this XML course was that I missed the first day of the expo, including the keynotes and all the seminars. Initially, I was a bit con-

cerned about this, but unless day one was very different than the next two, I don't believe I missed too much. While I learned a lot of specific things on the Wednesday and Thursday of the expo, a common thread ran through the keynotes and seminars. Summed up, it might read like this: Web services holds a lot of promise, but there are vast areas that remain undefined. I'm getting a little ahead of myself here, so bear with me awhile and we'll come back to this point.

My original question was: Is the concept of Web services just hype? The simple fact that this conference existed inclined me to believe that there had to be something more. The dozens of vendors I mentioned earlier spent big money developing supporting products. While it is possible that they are merely chasing the hype machine, I sense otherwise. Please don't get me wrong. I'm in no position to declare Web services the be-all and end-all. But there is something there.

Eric Newcomer of IONA gave a very powerful keynote on Wednesday entitled, "Web Services: Integration Technology for the 21st Century." He led with the statement that there are currently too many definitions of precisely what Web services is. While this may be strictly true, a fairly standard definition emerged during Newcomer's speech and in all the other talks I attended. Web services deals with the space between applications. It is a service-oriented architecture using XML and XML-based standards to identify,

seminars also had a Web services tilt.

In addition to an expo floor with dozens of vendors offering products and services either supporting Web services or claiming actually to enable them, there were 60 hour-long seminars divided among five tracks – Java, Web Services, XML, IT Strategy, and .NET. On top of all this, Sun ran six day-long courses from their Java University

AUTHOR BIO

Steven Berkowitz has done development and project management for Fortune 100 companies, startups, and nonprofit organizations.



describe, define, and execute business processes. Newcomer's holy grail is the built-on-the-fly business applications assembled from pluggable Web services sitting out there on the Web ready to go.

Now, this new world of interoperability will have monetary benefits. Without increased ROI, why would our bosses want to spend money on this stuff? Make no mistake, they will be spending the money. A slide from the "Practical Experiences with Web Services and J2EE" seminar quotes surveys from groups like Gartner, Jupiter Media Metrix, and InfoWorld showing that money is heading this way.

One of the primary cost savings with Web services, according to Newcomer, is that it will deskill programmers. We won't need mAd SkILz to integrate applications anymore. Business users can just pick from a menu of functionality and, voilà, a new application. We as developers may bristle at that; you may say that as a developer myself, I'm in outraged denial. But, this review will look at the undefined portions of Web services and the competing standards under development that are meant to address them. Look at that and then tell me how your accounting department is going to make their own applications Web services-enabled. We will still be needed. There really is no alternative.

Another problem with Newcomer's pluggable application vision is that we are nowhere near it. When talking about companies already using Web services, speakers repeatedly coughed up phrases like "most" and "90%" and "the vast majority" in front of "are using it for internal integration rather than exposing business processes to their partners."

That this is the case makes sense if you look at what Web services covers and, more tellingly, what it does not.

What everybody seems to agree on is that Web services is currently defined by three core standards: the Simple Object Access Protocol (SOAP), the Web Services Description Language (WSDL) and Universal Discovery, Description and Integration (UDDI). WSDL is based on XML Schema and a WSDL document is an XML document that defines and describes a service. A SOAP message is an XML document with

for prime time: management, security, context, delivery, workflow, choreography, service agreements, service characters, discovering, business system registration, and standards and conventions. That's a pretty hefty list of



the payload, delivering the data for the service to operate on. And UDDI is the yellow pages of Web services. It tells us where to find the services we need. While SOAP and WSDL are not yet W3C standards, they have become de facto standards and soon will be official.

Sounds great. It is, in a lot of ways. Remember that most companies are still using Web services for internal integration projects. The reality is, most enterprises have disparate systems – legacy, J2EE, Microsoft – all of which need to speak with each other. If you give them each a way to write SOAP messages for their outgoing information and read SOAP messages for

things in the way of widespread adoption. There are standards bodies addressing these things: W3C, Web Services Interoperability Organization (WS-I), and Organization for the Advancement of Structured Information Systems (OASIS) are just three. OASIS has 30 tech committees alone working on Web services issues.

Manes wasn't the only one to make the point about competing standards. Even "Practical Experiences with Web Services and J2EE," by Hugh Grant of Cape Clear, stressed these issues. The J2EE thrust of that hour was that you can translate your EJBs into Web services using WSDL to describe them, and a Façade pattern to cover multiple beans if they combine to a single service. Beyond that, his talk centered around the standards, which I found very telling.

I can give you the alphabet soup of proposed standards, but I think that



incoming data, you've made them talk with each other. There is value in that.

However, adoption of interenterprise Web services is hampered when you consider what is not defined by Web services. Little things like security, workflow, and transaction support. Web services currently does not address these issues in a standard way. Anne Thomas Manes of Systinet gave a talk on "Developments in Web Services Standards." Manes said that Web services needs to address the following issues before it can be truly ready

would cloud the real issue. You have to realize that many of the standards under development compete with one another. As we know, multiple standards to address a single issue mean there is no standardization. This is a very risky place to be, because it allows vendors to fill in the gaps. This is where Web services is now. There are too many holes, and vendors, such as Actional, SilverStream, and Sonic, all have products that will come in and fill many of those gaps. Many other



J2ME



J2SE



J2EE



Home

Report

web services
Conference & Expo
EDGE

vendors offer more tailored solutions, some addressing security, others workflow.

Phillip Merrick of webMethods gave a keynote entitled "Enterprise Web Services: An Evolutionary View of Web Services." He made a crucial point – the history of many technological issues is this: proprietary solutions (immature), proprietary solutions (mature), standardized solutions (immature). From here, there is a crossroad. We can either go to standards (mature), which would be ideal, or standards plus proprietary adds from vendors, which would be bad. It would fracture the Web services market and more or less kill the interoperability promise.

Web services is at this crossroad. If what I saw on the expo floor is any indication, we are going down the bad road. If you don't want to see that, take Merrick's advice. Demand standardization. Keep your vendors honest. If we can do that, the promise of Web services can be met. And that would be a fine thing indeed.

Even in the face of all the seminars and keynotes, I managed to spend some time on the expo floor, speaking with the vendors, learning a bit about what they had to offer. The two products I found most fascinating deal with XML in general, giving them a broader applicability than the Web services-focused products.

First, there is Allora from HiT Software. Coming in winAllora (COM) and jAllora (Java) flavors, this is a tool for mapping XML to relational data. Even my limited dabbling in XML showed just how different these beasts can be. While it is easy enough to take a JDBC ResultSet and write it all to an XML document, I can see that this would get very tedious for more than the odd table or two. The Graphical Mapper shows your XML Schema or DTD on one side and your database schema on the other, allowing you to map your relationships at design time. Plus, Allora comes with mapping, data-binding, and messaging APIs to make it useful at runtime. These two features alone might be worth the price of admission, but Allora goes one step further – it includes a set of SOAP interfaces so you can expose it as a Web service. The holiday weekend and publishing schedules have kept me from

installing and running it at home, but I was very impressed with the hands-on demo of the Graphical Mapper.

The other product that caught my fancy was Forum System's XML Web Services Security Appliance, which deals specifically with securing XML documents. This is a hardware device that plugs into your network between the firewall and the application server. As we know, XML is basically self-

pages, Alan Williamson. Because this conference had a .NET track (which, admittedly, I did not take advantage of), I had hoped for a fairly well-reasoned discussion. Some good points were made on both sides. Sadly, most good points were buried in bickering and the entire discussion was peppered with not-so-thinly-veiled snipes between the two camps. Maybe it was naïve of me to expect more, maybe it would have been

better to leave representatives of Microsoft and Sun out of the discussion, but as it stood, the whole thing left a sour taste in my mouth.

Finally, right next to the Web Services Edge was PC EXPO/TECHXNY and attendees of one got admission to

the other. While this may seem like a great deal, free admission to PC EXPO after all, it really took away from the power of Web Services Edge. At the risk of sound horribly I33t, the net effect was hordes of mere users, wearing "Gimme Free Stuff" buttons, gawping like tourists at booths displaying things they wouldn't understand even if they wanted to. They clogged the aisles, wasted exhibitors' time, and in some



describing data. An element called "password" or "card number" kind of screams out to the world that

there's good juicy data there. This is rather a gaping vulnerability that the security appliance addresses by encrypting the data, either in totality or on an element-by-element basis, and protects it from end-to-end.

"In addition to an expo floor with dozens of vendors offering products and services either supporting Web services or claiming actually to enable them, there were 60 hour-long seminars divided among five tracks – Java, Web Services, XML, IT Strategy, and .NET"

I would be remiss if, in the focus on Web services, I failed to point out that the conference also had a Java focus. The best Java presentation I went to was given by Rick Hightower of Trivera. His talk on "Java Tools for Extreme Programming" was well presented and rather informative. Rick focused on the testing and continuous integration portions of XP, discussing JUnit, Ant, and Cactus in detail. His talk proved to me that no matter what you are doing, no matter what you call your methodology, testing is invaluable and that these tools will make your life easier.

On Thursday there was a panel discussion on .NET versus J2EE. The stars were Simon Phipps, chief technical evangelist at Sun, and Barry Gregory, principal technology specialist from Microsoft. Sharing the stage with them were Sean Rhody, editor-in-chief of *Web Services Journal*, Mike Kovach of FullTilt Systems, and the editor-in-chief of these very

cases kept us Web Services Edge attendees from speaking to the vendors. What's more they glommed up more than their share of the goodies. And let's face it, to tech folk like us, corporate toys are second in importance only to free food.

Summing up: Web services exists in some form. It is coming. You can either help define it or live with what happens out there. But if you're reading this magazine, you'll end up working with it on some level. Whatever shape this mess takes, one thing is clear – XML is the lingua franca of Web services. Learn it. Live it. Love it. I know I spent a lot of time talking about what Web services isn't. Sometimes, though, that's more important than learning what something is. Which made my four days at the Web Services Edge Conference & Expo invaluable. ☘

sjb47@yahoo.com

motorola
www.motorola.com

Programming with a Model T

Vi and Emacs were good editors in the time before there was light (“Integrating Development,” by Ajit Sagar [Vol. 7, issue 6]). Punch cards were good too, but I don’t see anybody using them. If you’re using vi and Emacs, you’re programming with a Model T at Model T speed in an era of faster-than-light machines with 1TB of L1 memory! Get yourself a good IDE, Borland JBuilder, or whatever and reprogram yourself to its functions.

Greg Brett
greg.brett@faa.gov



C# – Premature and Incomplete

The problem with C# is that I strongly suspect it’s vastly overrated by uSoft (“There May Be Trouble Ahead...” by Alan Williamson [Vol. 7, issue 4]), but software managers and team leaders will choose it over Java because:

1. It appears easier and more productive to use. uSoft will include ease-of-use features and wizards in its IDE to promote this illusion.
2. No one ever got fired for choosing uSoft.

C# is a premature, incomplete, but seductive product. Anyone truly serious in opposing this technology, however, must learn C#, and use it to build sufficiently complex apps to discover its shortcomings. Then, when it’s your turn to contribute to a Java-versus-C# decision in your company, you’ll have real hard evidence. It’s unlikely your opponent will know Java well, so you’ll have the upper hand. You may even save your project from failure. You may even push Java to incorporate some improvements from C#, of which there are undoubtedly a few.

Simply, knowledge is power.

Peter Elgee
pelgee@videotron.ca



ed on Sun’s developer forum and on the JavaLobby forum, called “Inter-EJB calls 2,000 times faster by using local EJBs – great! But how?”

It’s important and difficult to avoid read/write static fields with respect to Singletons.

Thomas Taeger
taeger@classic-and-class.com

Correction to References

My article in *JDJ*, “Java Design: Creating Flexible Code,” (Vol. 7, issue 6) used concepts and content from a number of



JavaWorld articles without proper attribution. The articles in question are listed below as references, as they ought to have been when the article originally appeared. I apologize to the author Wm. Paul

Rogers, as well as to the readers and staff of JavaWorld and *JDJ*. This was a careless error on my part.

References

- (Image and code) “Reveal the magic behind subtype polymorphism”: www.javaworld.com/javaworld/jw-04-2001/jw-0413-poly-morph.html
- “A primordial interface?”: www.javaworld.com/javaworld/jw-03-2001/jw-0309-primordial.html
- “Thanks type and gentle class”: www.javaworld.com/javaworld/jw-01-2001/jw-0119-type.html

I would also like to note that the *Java Language Specification* and *Java in a Nutshell* by Flanagan (O’Reilly) was used to write this article. The Data Access Object pattern is based on the pattern as discussed in the Sun Blueprints and *Core J2EE Patterns: Best Practices and Design Strategies* by Alur, Crupi, and Malks (Prentice Hall PTR).

Mike Barlotta
mike.barlotta@aegis.net

Java Apps on Mobile Phones

While a Java app on your phone may not be able to make or answer phone calls, it should at least be able to make use of an Internet connection (“My Kingdom for a Phone,” by Alan Williamson [Vol. 7, issue 5]). I guess most applications would rather use

Internet services than call other people anyway. In any case, I would only want such a feature enabled for trustworthy programs. Just imagine, a virus that infected millions of phones and called stored numbers at random, or even a rogue program that could call your contacts and

impersonate your voice.

Alex Prayle
alex@zookitec.com



The Pulse of the Tech Professional

“Should I Stay or Should I Go?” by Bill Baloglu and Billy Palmieri (Vol. 7, issue 7) is a wonderful and humorous article. The authors’ fingers are really on the pulse of what goes on inside the mind and heart of the tech professional!! Bravo!!!

Lola
Diamondgirl0519@aol.com

The Perfect Answer

“Programming Restrictions in EJB Development” by Leander van Rooijen (Vol. 7, issue 7) is the perfect answer to my problem with a local EJB that I post-



Good Insight

“The Critical Role of Application Architecture” by Walter Hurst (Vol. 7, issue 5) provides good insight into database programming in an enterprise application. It

gives crucial inputs that help design scalable and reliable architecture.

Vijay Raghavan
vijayck_raghavan@hotmail.com



sys-con media
www.sys-con.com

Making It Useful



WRITTEN BY
BILL BALOGLU &
BILLY PALMIERI

But that love can also overflow and drown out little details like business plans and common sense. In the past several years, technological advances leapfrogged over real-world needs.

As a result, a lot of companies and highly creative, motivated engineers expended a tremendous amount of time, effort, and money creating “killer apps” that had no real problems to solve.

A typical example of this is a company that developed a supply chain application for the apparel industry. What the developers failed to realize is that the apparel industry at the time was running quite nicely via fax and paper.

Many of the intended customers for this high-tech application are small companies run by non-English-speaking people. For this application to work in the real world, its developers needed to convince everyone in the industry to (1) decide on a common language, and (2) buy into – and learn – one system. Guess what. It didn't happen.

It was once easy to find talented engineers to buy into these pie-in-the-sky enterprises, with the lure of untold billions once the product hit the street. But a lot of the senior engineers we talk to today now have a different set of priorities. They no longer ask, “How much stock are they offering and when is the IPO?”

Today's top engineers are no longer interested in building technology for its own sake; they want to build something that will solve real problems. Many of them tell us that they became engineers to make something useful. However, after months or years of hard work, many have found they've created something that's useless.

One of the distinctive qualities of the high-tech industry is that the people who make technology love technology. They love its powerful capabilities, its potential, and the opportunity it provides for creativity.

Chris has been a software architect for the past 10 years. His professional experience spans both startup and major corporate environments. For the past year and a half, he's worked at a major “household name” software company where teams of up to 50 engineers build his software applications.

“People who write software are creative,” he says. “But on a large software project with teams of 10 or 20 people, not everyone can create their own error-handling mechanism.

“Lots of engineers come into a major project and think they can do it better, so they go off and do their own thing,” says Chris. “The architectural discipline is important. There are lots of aspects that go into the product.”

In his experience, many engineers lack the ability to follow an architectural guideline. He attributes this problem to the current focus of software engineering education.

“They spend a lot of time teaching programming languages, so everyone focuses on coding,” he says. “But almost no one who's come out of school recently knows what it means to build a product or understands the value of architecture.”

Recent engineering graduates entering the corporate world have an understandable need to prove themselves. According to Chris, many of them try to establish themselves by doing things their own way, trying to prove that they can do it better, faster.

“The startup culture encouraged that approach,” he says. “Many had very loose, creative environments with people becoming heroes by working 24 hours straight over the weekend.”

Solving real-world problems

That culture also nurtured the attitude of “let's make it and then see if it works,” which is the complete opposite of what's required when a major company sets out to design and build a product.

Many of these creative, lone-gun engineering heroes now have difficulty working in a collaborate environment, on a specific piece of a very large puzzle.

“From the company's viewpoint it creates issues,” he says. “All engineers on the team need to be clear on the product and the architecture. They need to know the technology and the alternatives, to understand the whole product, not just one part of it.”

Chris observes that with the startup boom behind us, the interview process has become more effective – to a point. “People are looking for very specific skillsets now; they're not just hiring anyone who's available.”

Yet the hiring process doesn't always screen out candidates who may be temperamentally unsuited for a large-scale project when the interview consists only of a candidate talking with a couple of people for two hours.

Many of the major companies have implemented more comprehensive, rigorous hiring processes to screen candidates not only for skillsets and technical expertise, but for their ability to work effectively in a team.

Because when creating major products to solve real-world problems, there's a lot more required than creativity – and heroism.

• • •

E-mail us your feedback. We're always interested in hearing from you. ☛

jdcolumn@objectfocus.com

ADVERTISER	URL	PHONE	PAGE
Actuate Corporation	www.actuate.com/info/fbjad.asp	800-884-8665	51
Altova	www.altova.com		61
AltoWeb	www.altoweb.com		21
AppDev Training	www.appdev.com	800-578-2062	58
Apple Computer, Inc.	www.apple.com/xserve		4-5
BEA	dev2dev.bea.com/useworkshop		6
Borland Software Corp.	www.borland.com/new/jb7/5068.html	800-252-5547	29
Canoo Engineering AG	www.canoo.com/ulc/	41 61 228 94 44	25
Capella University	www.capella.edu/getcred	1-888-CAPELLA ext. 6027	32
Compuware Corp.	www.compuware.com/products/optimalj	800-468-6342	11
Covasoft	www.covasoft.com/today's_tech/		103
CTIA Wireless I.T. & Internet 2002	www.ctiashow.com		63
Dice	www.dice.com		41
Engenuity Technologies	www.jloux.com	800-684-5669	47
ESRI	www.esri.com/arcims	888-289-5084	33
Fiorano Software	www.fiorano.com	800-663-3621	53
IBM	ibm.com/wesphere/ebaydev		27
InetSoft Technology Corp.	www.inetsoft.com/jdj	888-216-2353	55
Infragistics, Inc.	www.infragistics.com	800-231-8588	14-15
INT, Inc.	www.int.com	713-975-7434	34
Java Developer's Journal	www.sys-con.com/java		75
Jinfonet	www.jinfonet.com/jdj8.htm	301-838-5560	31
Macromedia	www.macromedia.com/go/jrun4jdj		37
Metrowerks Corp.	www.wireless-studio.com		8
Motorola	www.motorola.com/developers/wireless		73
n software inc.	www.nsoftware.com		49
n-ary	www.javaSOS.com		43
Northwoods Software	www.nwoods.com/go/	800-434-9820	30
Oracle Corp.	www.oracle.com/ad	800-633-1072	17
Parasoft Corporation	www.parasoft.com/jdj8	888-305-0041	35
Precise Software	www.precise.com/jdj	800-310-4777	23
QUALCOMM Incorporated	http://brew.qualcomm.com/ZJD5		65
Rational Software	www.rational.com/offer/javacd2		13
SilverStream Software	www.silverstream.com/coins	1-888-823-9700	39
Sitraka	www.sitraka.com/jclass/jdj	800-663-4723	19
Sitraka	www.sitraka.com/jprobe/jdj	800-663-4723	57
Sitraka	www.sitraka.com/performance/jdj	800-663-4723	104
Sonic Software	www.sonicsoftware.com/jdj	800-989-3773	2
Sprint PCS	http://developer.sprintpcs.com		67
Zero G	www.zerog.com	415-512-7771	3, 59

General Conditions: The Publisher reserves the right to refuse any advertising not meeting the standards that are set to protect the high editorial quality of *Java Developer's Journal*. All advertising is subject to approval by the Publisher. The Publisher assumes no liability for any costs or damages incurred if for any reason the Publisher fails to publish an advertisement. In no event shall the Publisher be liable for any costs or damages in excess of the cost of the advertisement as a result of a mistake in the advertisement or for any other reason. The Advertiser is fully responsible for all financial liability and terms of the contract executed by the agents or agencies who are acting on behalf of the Advertiser. Conditions set in this document (except the rates) are subject to change by the Publisher without notice. No conditions other than those set forth in this "General Conditions Document" shall be binding upon the Publisher. Advertisers (and their agencies) are fully responsible for the content of their advertisements printed in *Java Developer's Journal*. Advertisements are to be printed at the discretion of the Publisher. This discretion includes the positioning of the advertisement, except for "preferred positions" described in the rate table. Cancellations and changes to advertisements must be made in writing before the closing date. "Publisher" in this "General Conditions Document" refers to SYS-CON Publications, Inc.

What's in the next issue of *JDJ*?

MANAGING JAVA SOURCE CODE DEPENDENCIES FOR SCM



This article explores the evolution of the typical Java source code tree and articulates the underlying relationships that make even basic Java SCM problematic. It also suggests a simple way to manage source code relationships to meet basic SCM goals.

THE ANATOMY AND PHYSIOLOGY OF EJB 2.0 PRIMARY KEYS

When implementing container-managed persistence in EJB 2.0, what are some of the decisions and details involved? In an evolving marketplace, business requirements keep changing, so something as important as the Primary Key of an entity should be as constant as possible.

CREATING A CUSTOM LAUNCHER

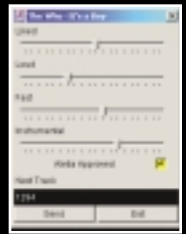
The most frustrating and error-prone aspect of Java for the average user is just starting a Java program. The monumental confusion of batch files, scripts, and command-line cut-and-paste necessary to start a Java program using the default launcher is an ongoing problem area even for veteran developers. This article shows how you can wipe away the whole mess and easily write custom launchers for your applications.

WHOLE HOUSE AUDIO IN THE PALM OF YOUR HAND - PART 2

Part 1 showed you how to develop an MP3 player in Java, then control that player from a wireless handheld device using a PersonalJava application. However, you can get some wild combinations when listening to your entire music collection at random. When a nice relaxing Enya track fades out and you find yourself launching into the world of Eminem, the shock can be considerable. Learn how to solve this problem as well as provide an ideal wireless mechanism for this application.

TRAINING DAYS

It's no secret to anyone who works in the technology industry that continued training (and retraining) is required on an ongoing basis. The only thing that's constant in this business is change, and engineers need to be ahead of the curve on the latest and greatest technologies.



How I Spent My Summer Vacation



WRITTEN BY
BLAIR WYMAN

I almost missed this month's deadline for this column, but I have a good excuse: I've been out searching for beauty again, and must report that my family and I found some largish, mountain-shaped accumulations of it in Glacier National Park, in north-western Montana.

Now, the pursuit of this beauty didn't come cheap – in 10 days, we semi-consciously put over 3,000 miles on the trusty minivan – but the relative drudgery of the to and fro was surely and stupendously compensated by the bald-faced bundles of blatant beauty we encountered when we got there.

The trip to the far side of Montana, clear across four of the larger states, was mostly as uneventful as the vast interstate highway system could possibly promise: no flat tires, no breakdowns, and no vast herds of unruly or uneducated bison trampling my precious life-sustaining box of little chocolate donuts. (Give me enough strong black coffee and little chocolate donuts, and I'll drive around the world.)

The familiar trip from Minnesota to Wyoming (via I-90) crosses over 400 miles of my home state of South Dakota, and is remarkable (if a little painful) for its scenic monotony. With notable exceptions (and given a few pasture-sized bicubic patches), nearly any stretch of the trip could be inconspicuously replaced with any other.

On the bright side, trips like this can deepen your appreciation of the world's vastness. (Yeah, I actually tried that one on my kids.) On the dark side, when the charm of vastness wears thin, the horizon starts to look like a looping cartoon background: Tom chases Jerry across the apparently infinite, hay-carpeted

living room, passing that same cheesy billboard every 950ms. Been there. Done that. It's really big and mostly flat.

To be fair, the broadly dispersed notables on the trans-Dakota trip include such gems as the reservoir behind the dammed Missouri River at Chamberlain, those darned Badlands near Wall, and the doggoned Black Hills of South Dakota. Now, each of these locations has already been a destination in my search for beauty (and, hopefully, will be again), but this summer the ultimate destination was the mountains – big mountains.

The Black Hills of South Dakota are sort of like mountains. As geologic features go, I've been told the "black mountain hills of Dakota" are among the oldest crustal features around – even older than the Beatles' song that made them famous – and that this age accounts for the mostly smooth contours of their weathered granite and pine-covered swells. Swells are swell, I guess, but give me a good rocky crag or two this summer!

Less geologically, more colloquially, the notable sights on I-90 include several concrete dinosaurs, a large metal pasture sculpture of a bull head, and a billboard offering 24-hour "toe" service. With such visual wealth, how could you want for more? Well, if you're bold enough, or the gas gauge approaches "E" at the right time, you might want to take a short side trip to Mitchell, South Dakota, and pass

one of the strangest buildings on earth.

Forming the outer layer of this oddly shaped building, you'll find literally thousands of carefully mutilated carcasses – ritualistically dismembered, drawn and quartered, and finally stapled to sheets of plywood. These carcasses are arranged according to their primary genetic attributes of color and morphology, in such a fashion as to form patriotic images of pioneers, astronauts, firefighters, farmers, nuns, and such like.

Okay, well, I guess "carcass" is not the best choice of words. But, if ears of corn *were* sentient, then the so-called "Corn Palace" of Mitchell, South Dakota, would surely be (au)reality's single unholy abomination.

Now, the trip west of Dakota – into Wyoming and Montana – will have to be the stuff of some future article. I'm back at work now, but I'm not the same as I was before the trip. Summer vacation melts all the year's frozen goo in my head and then pulls my cranial drain plug, so that the molten goo runs out of my ears and directly onto my T-shirt. (To the naked eye, the goo stain looks just like spilled coffee.)

If you happen to see me at some midwestern truck stop next summer, look for the stain. If it's there, the trip is going well. If it's not, don't worry: it's just a matter of time. ☘

blair@blairwyman.com



covasoft.com

sitraka
www.sitraka.com