

ADVERTISER INDEX

ADVERTISER	URL	PH	PG
4TH PASS	WWW.4THPASS.COM	877-484-7277	89
ADRENALINE	WWW.ADRENALINE.COM		12
ADVANCED DATA, INC.	WWW.JVSEARCH.COM		46
AFFINITY	WWW.AFFINITY.COM/RESELLER	800-646-0662	41
AIR2WEB	WWW.AIR2WEB.COM	404-815-7707	25
AVANTSOFT	WWW.AVANTSOFT.COM/SEMINAR	408-530-8705	40
BEA	DEVELOPER.BEA.COM	408-570-8000	17
CAPE CLEAR	WWW.CAPECLEAR.COM		33
CAREER OPPORTUNITY ADVERTISERS		800-846-7591	114-137
CEREBELLUM SOFTWARE	WWW.CEREBELLUMSOFT.COM	888-862-9898	63
CODEMARKET	WWW.CODEMARKET.COM	646-486-7346	99
COMPUTERWORK.COM	WWW.COMPUTERWORK.COM	800-691-8413	52
COMPUWARE	WWW.COMPUWARE.COM/NUMEGA	800-4-NUMEGA	35
CORDA TECHNOLOGIES	WWW.CORDA.COM	802-802-0800	69
CTIA WIRELESS IT 2000	WWW.WIRELESSIT.COM	202-736-3241	105
CYSCAPE	WWW.CYSCAPE.COM/BHFREE	800-932-6869	52
ELIXIR TECHNOLOGY	WWW.ELIXIRTECH.COM/DOWNLOAD	65 532-4300	107
ENSEMBLE SYSTEMS	WWW.ENSEMBLE-SYSTEMS.COM	877-290-2662	38
ESMERTEC	WWW.ESMERTEC.COM		27
FATBRAIN.COM	WWW.FATBRAIN.COM/FBT/OFFERS/JAFADEV		21
FIORANO	WWW.FIORANO.COM	800-663-3621	81
FLASHLINE.COM, INC.	WWW.FLASHLINE.COM	800-259-1961	67
FORT POINT PARTNERS, INC.	WWW.FORTPOINT.COM	415-395-4400	19
GEMSTONE	WWW.GEMSTONE.COM	503-533-3000	57
GENERIC LOGIC, INC.	WWW.GENLOGIC.COM	413-253-7491	89
HIT SOFTWARE	WWW.HIT.COM	408-345-4001	73
IAM CONSULTING	WWW.IAMX.COM	212-580-2700	85
INETSOFTECHNOLOGY CORP	WWW.INETSOFTECHNOLOGY.COM	908-755-0200	75
INSTANTIATIONS, INC.	WWW.INSTANTIATIONS.COM	800-808-3737	87
INT	WWW.INT.COM	713-975-7434	22
INTERNET WORLD FALL 2000	WWW.PENTONEVENTS.COM	800-500-1959	112-113
INTUITIVE SYSTEMS, INC	WWW.OPTIMIZEIT.COM	408-245-8540	101
JAVA DEVELOPER'S JOURNAL	WWW.JAVALDEVELOPERSJOURNAL	201-802-3020	58,111
KL GROUP INC	WWW.KLGROUP.COM/GREAT	888-361-3264	15
KL GROUP INC	WWW.KLGROUP.COM/FASTER	888-361-3264	53
KL GROUP INC	WWW.KLGROUP.COM/MASTER	888-361-3264	95
KL GROUP INC	WWW.KLGROUP.COM/DEPLOY	800-663-4723	140
NO MAGIC	WWW.MAGICDRAW.COM	303-914-8074	7
NORTHWOODS SOFTWARE CORPORATION	WWW.NWOODS.COM	800-226-4662	102
OOP.COM	WWW.OOP.COM	877-667-6070	37
PACIFIC NORTHWEST NATIONAL LABORATORY	WWW.PNNL-SOFTWARE.COM	509-946-1110	47
PARASOFT	WWW.PARASOFT.COM/JDJ_OCT.HTM		61
POINTBASE	WWW.POINTBASE.COM/JDJ	877-238-8798	43
PRAMATI	WWW.PRAMATI.COM	408-965-5513	83
PROGRAMMER'S PARADISE	PROGRAMMERSPARADISE.COM/VISUALSOFT	800-516-0101	29
PROGRESS SOFTWARE	WWW.SONICMQ.COM/AD11.HTM	800-989-3773	2
PROTOVIEW	WWW.PROTOVIEW.COM	800-231-8588	3,97
QUADBASE	WWW.QUADBASE.COM	408-982-0835	45
QUICKSTREAM SOFTWARE	WWW.QUICKSTREAM.COM	888-769-9898	109
SANDSTONE TECHNOLOGY	WWW.SAND-STONE.COM	858-454-9404	40
SEGUE SOFTWARE	WWW.SEGUE.COM	800-287-1329	11,13
SIC CORPORATION	WWW.SIC21.COM	822.227.398801	91
SIERRA SYSTEMS	WWW.SIERRASYSTEMS.COM		59
SLANGSOFT	WWW.SLANGSOFT.COM	972-2-648-2424	139
SOFTWARE AG	WWW.SOFTWAREAG.COM/TAMINO	925-472-4900	79
SOFTWARED	WWW.SOFTWARED-INC.COM	41-14452370	65
THESHORTLIST	WWW.THESHORTLIST.COM	877-482-8827	39,47
THINAIRAPPS	WWW.THINAIRAPPS.COM/JAVA	888-609-THIN	49
TIDESTONE TECHNOLOGIES	WWW.TIDESTONE.COM	913-851-2200	77
TOGETHERSOFT CORPORATION	WWW.TOGETHERSOFT.COM	919-833-5550	6
UNIFY CORPORATION	WWW.UNIFYEVAVE.COM	800-GO UNIFY	93
VERGE TECHNOLOGIES GROUP, INC.	WWW.EJIP.NET	303-998-1027	103
VERIO	WWW.DEDICATEDSERVER.COM	877-624-7897	23
VIRTUALSCAPE	WWW.VIRTUALSCAPE.COM	877-VSCAPE4	31
VISICOMP	WWW.VISICOMP.COM/JDJ10	831-335-1820	51
VISUALIZE INC.	WWW.VISUALIZEINC.COM/JDJ	602-861-0999	58
WEBGAIN	WWW.WEBGAIN.COM	408-517-3815	4,55

Please visit
the Web sites
of our
advertising
partners
who make it
possible for us
to bring you this
Digital Edition
(PDF) of *JDJ*



JAVA™ DEVELOPER'S JOURNAL

The World's Leading Java Resource

November 2000 Volume:5 Issue: 11

JAVADEVELOPERSJOURNAL.COM



CORBA Corner

by Jon Siegel pg. 16

Industry Watch

by Alan Williamson pg. 30

Interviews

Bill Baloglu

of ObjectFocus Inc. pg. 90

Ted Farrell

of WebGain pg. 100

RETAILERS PLEASE DISPLAY UNTIL JANUARY 31, 2001 \$4.99US \$6.99CAN



by Francisco Morales page 48

Extending Java's concurrency model with asynchronous objects

Eiffel-Like SEPARATE CLASSES



Feature: Pluggable Session Management for Java Servlet-Based Web Applications

Setting standards for the Web site developer



Viswanath Ramachandran 8

Feature: Resource Pooling in Java

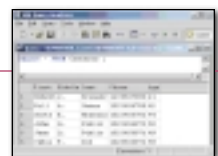
Produce a resource pool for production-level heavy lifting



Ivan Kiselev 22

Feature: Logging

Third-party logging API helps trace and debug problems



Vinay Aggarwal 34

Java Basics: After the Connection

To understand JDBC, first get to know SQL

Robert J. Brunner 42

EJB Home: Extreme Programming with EJB

Under the right conditions, EJB and XP are a powerful combination



Jason Westra 58

Feature: Java Tools for Embedded Systems

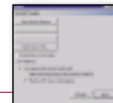
Providing Java support in the embedded space



Sherry Shavor, Peter Hagggar, Greg Bollella 64

Java Techniques: Persistence Frameworks

Save time and money and improve your database apps



Mike Jasnowski 84

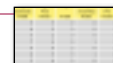
VAR: Create XML Based Web Applications

XML and Java tools make development and debugging easy

Luc Chamberland Arthur Ryman 92

Debugging: Automated Software Inspection

Save development costs and improve reliability



Scott Trappe Lawrence Markosian 102



SEAN RHODY, EDITOR-IN-CHIEF



Configure This!

There's a project team out there that really hates me. I was called in recently to help them get their application together so it could be put into production. When I got there, I determined that the problem was simple – no one understood configuration management.

It's tempting to say that in the old days configuration management was easy. You just created your executable and distributed it to your clients, and away you went. It's tempting, but that's a load of baloney. Configuration management was an issue on the mainframe, it was an issue on client/server, and it's still an issue today. Any application of any substance is created by a team of developers, and includes configuration files, multiple-source code files, and a host of other options, all of which contribute to the complexity of software development.

What has changed is that the complexities of the systems we're building have increased while the skill of the average programmer, by definition, has remained constant. The number of paradigm shifts that have occurred in the past decade has made it difficult for the average programmer to even understand *where* code will be deployed, let alone *how*.

I'm not speaking about small, one- or two-developer projects, although even there the scope of the configuration process can get away from you. I'm concerned more about the large-scale, multi-million-dollar development efforts aimed at putting a high-volume transactional site up on the Web. With the rise of clustering solutions, Network Attached Storage, redundant hardware, and distributed software, it's become much more challenging to deploy an application or even to debug a problem.

Let's consider a simple example. Suppose we're building a transaction system based on Enterprise JavaBeans running in a clustered server. Now suppose we fix a flaw in the transaction bean, one in which a calculation was incorrect and was providing the wrong numbers. We don't change any interfaces; we just correct the logic. Then we deploy it, but by mistake we deploy it to only three of the four servers in the cluster. The fourth retains the old copy. Assuming round-robin scheduling and equal load, one out of every four invocations of the transaction will produce incorrect results. But we know we've fixed it – see, it works here on my machine.

That's the problem. "It works on my machine" isn't an acceptable answer. If you hear that on your project, chances are you're in trouble. It means that someone in your organization doesn't understand the complexities of the environment or the overall principles of configuration management.

For the most part, configuration management is an organizational task. It starts with source-code control. All developers should check in code every day. I was on a project once where a programmer lost six weeks of work because he hadn't checked it in and his disk crashed. That's a lot of recoding.

Proper identification of release materials is next. It's not enough to take the latest checked-in code (what if the developer is in the middle of a rewrite and the code doesn't work?). I've seen some projects try to get around this by having the developers provide the output, that is, the bytecode, instead of the source. This is also bad news. The team I worked with recently assured me that all their code was up to date, and that they were providing the output (in this case EJB JARs) because it was more efficient. I charged them a dollar for each missing file, or for each class that wouldn't compile. I made 50 bucks the first day on the compilation issues alone. Now you see why they hate me.

What's also important is to have a clear idea of what each environment looks like and map the distribution to it. Larger projects usually have several environments; there's one for development, one for testing, one for staging, and one for production. They don't always look the same. Sometimes they might not even have the same operating systems. It's not uncommon to see development take place on an NT server, but production takes places on a UNIX box. Java's portable, but one OS is case sensitive and the other isn't. It's important to understand the environment and to have a clear plan for deployment.

Hand in hand with the need for process is the need for automation. It's not enough to have a source-code repository. It's nice that it's got bug tracking now (most of the ones I've used lately, anyway), but we could do that in a spreadsheet. What most of the tools are lacking is a powerful tool that will make deploying code in sophisticated, disparate environments a simple, straightforward task. And the tools that do exist are in the wrong place. WebGain Studio has a great capability to deploy code into WebLogic. IBM's VisualAge does something similar for WebSphere.

That's nice, but for a number of reasons I don't want developers deploying code; it's not their job. It's the task of the configuration manager. That's why the proper place for deployment tools is in the repository. The repository is the basic tool of the configuration manager, and he or she needs that tool to be as robust as possible. It needs to go beyond source code management and defect tracking. These tools need to understand Java, EJB, JDBC, and a host of other technologies. We've needed them for a long time, but they're just starting to come on the market.

It's all about configuration management these days. Do it right and no one will know how good you are. Do it wrong and everyone will know. It's a dirty job, but someone's got to do it. Now go check in that file or you'll owe me a dollar. ☘

sean@sys-con.com
AUTHOR BIO

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

JAVA DEVELOPER'S JOURNAL

EDITORIAL ADVISORY BOARD

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

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

WRITERS IN THIS ISSUE

VINAY AGGARWAL, RUSLAN BELKIN, GREG BOLELLA, ROBERT J. BRUNNER,
LUC CHAMBERLAND, PETER HAGGAR, MIKE JASNOWSKI, DAVID JOHNSON,
IVAN KISELEV, LAWRENCE MARKOSIAN, JIM MILBERY, JOE MITCHKO,
FRANCISCO MORALES, VISWANATH RAMACHANDRAN, SEAN RHODY,
ARTHUR RYMAN, SHERRY SHAVOR, JON SIEGEL, SCOTT TRAPPE,
DON WALKER, JASON WESTRA, ALAN WILLIAMSON

SUBSCRIPTIONS

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

SUBSCRIPTION HOTLINE: 800 513-7111

COVER PRICE: \$4.99/ISSUE

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

PRESIDENT AND CEO: FUAT A. KIRCAALI
VP. PRODUCTION: JIM MORGAN
SENIOR VP. SALES & MARKETING: CARMEN GONZALEZ
VP. SALES & MARKETING: MILES SILVERMAN
ADVERTISING ACCOUNT MANAGERS: ROBYN FORMA
MEGAN RING
ASSOCIATE SALES MANAGERS: CARRIE GEBERT
CHRISTINE RUSSELL
ASSOCIATE ART DIRECTOR: DINA ROMANO
ASSISTANT ART DIRECTORS: LOUIS F. CUFFARI
CATHRYN BURAK
ABRAHAM ADDO
GRAPHIC DESIGNER: AARATHI VENKATARAMAN
GRAPHIC DESIGN INTERN: ROBERT DIAMOND
WEBMASTER: STEPHEN KILMURRAY
WEB DESIGNERS: GINA ALAYAN
SYS-CON EVENTS MANAGER: ANTHONY D. SPITZER

EDITORIAL OFFICES

SYS-CON PUBLICATIONS, INC.
135 CHESTNUT RIDGE ROAD, MONTVALE, NJ 07645
TELEPHONE: 201 802-3000 FAX: 201 782-9600
SUBSCRIBE @ SYS-CON.COM

JAVA DEVELOPER'S JOURNAL (ISSN# 1087-6944)
is published monthly (12 times a year) for \$49.00 by
SYS-CON Publications, Inc., 135 Chestnut Ridge Road, Montvale, NJ 07645.
Periodicals Postage rates are paid at
Montvale, NJ 07645 and additional mailing offices.
POSTMASTER: Send address changes to:
JAVA DEVELOPER'S JOURNAL, SYS-CON Publications, Inc.,
135 Chestnut Ridge Road, Montvale, NJ 07645.

© COPYRIGHT

Copyright © 2000 by SYS-CON Publications, Inc. All rights reserved.
No part of this publication may be reproduced or transmitted in any form or by any
means, electronic or mechanical, including photocopy or any information storage and
retrieval system, without written permission. For promotional reprints, contact reprint
coordinator, SYS-CON Publications, Inc., reserves the right to revise, republish and
authorize its readers to use the articles submitted for publication.

WORLDWIDE DISTRIBUTION BY

CURTIS CIRCULATION COMPANY

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

Java and Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc.,
in the United States and other countries. SYS-CON Publications, Inc., is independent of Sun
Microsystems, Inc. All brand and product names used on these pages are trade names,
service marks or trademarks of their respective companies.

BPA
SYS-CON
MEDIA



Letters to the Editor

The Online Solution

I am a Java software developer and just wanted to drop you a line to say how much I appreciate you putting past copies of your *JDJ* and *XML-J* online. It solves two problems for me: one, my printed copies usually go missing (they're such good reads), and two, if I missed a copy on the newsstand (which I do sometimes when I'm away), I can still access the old version. Being able to view last month's copy without having to go to the news agency is a real bonus, and I'll usually still buy the printed version if it covers the topics I'm interested in.

—Dr Brett Miller

brett.miller@marine.csiro.au

Download the Code

I just finished your excellent article on Analysis Patterns in the September '00 issue of the *Java Developer's Journal*. I am new to the Patterns space, so I am anxiously seeking code examples for as many patterns as I can find. Do you have some sample code of the "Observation Panel Bean" mentioned in your article?

—Jason Mawhinney

jason_mawhinney@payless.com

Ed. Note: The code listing for this and all articles in *JDJ* can be found at www.JavaDevelopersJournal.com.

JC2 Conference Kudos

My take on the show was that it was quite successful and really hit home with both developers and exhibitors. The feedback from developers and exhibitors was very positive. Developers found that the breadth, depth, and degree of technical coverage was very strong and, in most cases, exactly what they were looking for. A couple remarked that it was one of the best Java conferences they had attended. The exhibitors/vendors were all pleased with the exhibit hall/booth traffic and that the quality of leads and caliber of people they were speaking to was high.

—Joshua Duhl

jduhl@mediaone.net

I really enjoyed the conference and learned quite a bit. I wanted to make sure a few of the presenters received some credit.

Peter Hagggar's presentations were just great. He was very knowledgeable and had a down-to-earth presentation style. Peter was very adept at hopping between his presentation software, code samples in SlickEdit, and running code. His ability to run the examples out front of us really helped to bring some points home and make them more believable.

Martin Fowler is a must at any software development conference. He has an informa-

tive and entertaining presentation style. Martin did not want to be distracted with the laptop when speaking so he brought an associate to work the laptop. He also edited and compiled code out front of the group again making his concepts more tangible.

Ethan Hunt was incredibly knowledgeable. He was excellent. I learned a tremendous amount from his two presentations as well.

I would suggest the following for improvements:

- Presenters should be encouraged to run live code samples as part of their presentation to demonstrate their points. It's too easy to get up there and just read the slides to the audience.
- I was wondering why you bothered burning the presentations to a conference CD. I would suggest just making them available on your Web site. I would think this would make it easier for the presenters to make last-minute updates to their presentations and code samples.
- Thumbs down on transparencies regardless of what Learning Tree's "extensive study" came back with.

All in all, I really enjoyed the conference and felt it was well worth my while to travel cross-country for it.

—Steve Roy

s_d_roy@yahoo.com

Editorial Feedback

First let me commend you for the fairly balanced tone of your article ("The Sharp Tongue of Microsoft," *Industry Watch*, October)... In my opinion MS has made some very appropriate concessions to what you might call the pragmatic side of software engineering. Java has taken a rather extreme approach that in the long run will hurt it (my opinion).

Anyway, thanks for your objectivity. I am so tired of Java priests, acolytes, choirboys, zealots, etc. I guess, to extend my analogy, in the Java world I am a heretic...

Regarding customer service in the USA, I think it can be attributed, simply, to competition. We have a tradition of "limited government" and "freedom" (the freedom part we got from the British, of course). That tradition, which is actually the law as spelled out in our Constitution, is under fierce attack. Anyway, from an old TV advertisement, "When things compete, they get better."

Having said all that, most nonchild Yanks can remember when service was much better (in some areas). But the "consumer" mentality does come under attack, sometimes deservedly (for example, in discussions about how churches should operate – should they really have car washes, for example).

—Miles Whitener

No Magic p/u

Pluggable Session Java Servlet-Based

WRITTEN BY
RUSLAN BELKIN & VISWANATH RAMACHANDRAN

Numerous books and publications are available on the various technologies that support e-commerce on the Internet. As Java Servlets and JavaServer Pages (JSPs) emerge as a popular technology, a lot of material is being written about them. Most of this material focuses on programming model features, ease-of-development issues, and integration with tools. However, Web site developers are increasingly concerned about developing sites that can scale to a large number of hits while keeping the complexity of the software at a reasonable level.

This article focuses on how Web developers can utilize advanced features of the next generation of Web server products to design their server-side applications for scalability and performance using Java, and specifically discusses the state and session management aspect of Web applications. It indicates how the pluggable session management service of iPlanet Web Server can be used to achieve high performance and scalability. This article assumes you're familiar with the Java Servlet API.

URL Encoding and Cookies

The HTTP that's used to access almost all the information on the Internet is stateless. This presented a problem, especially for transac-

tion-based e-commerce sites, since they needed to identify a specific client across multiple requests. For example, when a client goes back to the server that's handling airline ticket reservations to confirm the selection, the server has to be able to identify the client and continue the transaction. One solution employed by early Web site developers was to encode relevant information directly into the response URL. This didn't require any additional capabilities from clients or servers, and it worked especially well on several prominent high-traffic sites. The method, known as URL encoding, is still widely used.

A later solution invented by Netscape for the state retention problem was the cookie mechanism. The server would send a cookie header to the client (browser), which would return that same cookie to the server on subsequent requests, thus allowing the server to uniquely identify the client. In addition, the cookie has a life span that allows it to expire. The cookie header, however, has a limitation on the length of the data it can transmit, and there's no mechanism enabling the server to ask for a specific cookie. Listing 1 provides the code for a Java Servlet that uses the cookie mechanism for maintaining the number of times a client has accessed the servlet.

Table 1 summarizes the features, advantages, and disadvantages of using cookies versus URL encoding.

Management for Web Applications



Server-Side Session Management

Due to size limitations of the URL encoding and cookie mechanisms, a need arose for more sophisticated session management techniques. Unfortunately, further revisions of the HTTP protocol elected not to address the issue of maintaining session data between the server and client, leading to the necessity of storing the session information associated with the client on the server side. Instead of transmitting the entire BLOB of data representing the session data (often not possible due to size limitations), the server would transmit a session cookie that contained only the session identifier (SID). In case the client didn't support cookies, the SID could be encoded in the URL instead. In the future, it would be used by the server to retrieve the session data stored somewhere on the server, typically in memory, on a disk, or in a database. The mechanism is generally referred to as server-side session management. The Java Servlet API provides programming model features that make the use of server-side session management extremely convenient and intuitive to the developer.

Table 2 summarizes the main features, and the advantages and disadvantages of the server-side maintenance of sessions versus the cookie-based mechanism.

Listing 2 contains the code for a Servlet that maintains a client session using server-side sessions.

From the application developer's point of view, managing the session with the client is simple. The Servlet container takes care of all the specifics for generating the session ID, identifying the incoming request, storing objects associated with the session, and managing the session life cycle. Practically all the application developer needs to do is call the `getSession ()` method on the `HttpServletRequest` object to obtain the reference to the session object.

	URL ENCODING	COOKIE
Size of session-specific data	Extremely limited by the size of the URL	Limited by the size of the header
Minimum client requirements	None	Client must support cookie headers
Performance/scalability effects	Practically none	Minimal. When using slow links, the fact that the client may send multiple cookies, thus increasing the amount of data transferred on each request, may affect the overall response time
Security risks	Information encoded in the URL is usually visible on the client	The cookie database usually stores cookies for multiple sites, thus posing small security risk due to buffer-overrun exploits found in older versions of clients

TABLE 1 Comparison of URL encoding vs cookies

	SERVER SESSION	COOKIE
Size of session-specific data	Unlimited	Limited by the size of the header
Client requirements	None (URL-encoded IDs can be used instead of cookies)	The client must support cookies
Security risks	Session IDs could potentially be guessed by the malicious client; cryptographically strong algorithms of SID generation must be employed to alleviate the problem	The cookie database usually stores cookies for multiple sites, thus posing small security risk due to buffer-overrun exploits found in older versions of clients
Performance/scalability effects	Server-side session management can pose significant scalability problems as it may need to maintain session information (sometimes distributed) across requests and instances	Large amount of data transmitted inside cookie headers may cause higher response times on slow links

TABLE 2 Comparison of server-side sessions vs cookies

Server-Side Session Properties

While the Servlet API offloads all the groundwork of managing the session with the client, making it easy to use, it doesn't specify anything about how session objects are created, where they're stored, what level of protection against failures is provided, and more. These issues are left for the Servlet container to define and implement. As we pointed out earlier, the session management could be inherently expensive and often dependent on the type and purpose of the Web application.

Depending on the type of application and specific customer requirements, you can examine the following considerations while evaluating the way the Servlet container manages sessions.

Life Span of the Session

Sessions can be short-lived (for a relatively short transaction, such as placing a stock trade). Usually these types of sessions are created and accessed often and you may want to make that process as lightweight as possible.

On the other hand, sessions can also be long-lived, for example, when the user logs into his or her brokerage account to perform various maintenance tasks and accesses pages at a relatively infrequent pace.

Persistence

The sessions may need to be made persistent – meaning the session data is stored in reliable storage and will persist there even across server restarts. By persisting your session you ensure a certain level of failover capabilities. Normally there'd be a price for that in terms of performance. The application developer must be aware that all objects placed into such a session must implement the `java.io.Serializable` interface.

Locality

When the server and the network setup can ensure that all requests from the same client will come back to the same server, there's no need to worry about the migration of sessions across server instances. However, a facility to access the session from the server instance that didn't create the original session may need to be provided. The simplest way to achieve this is to declare such sessions to be persistent and always store them in some central database.

More elaborate approaches can attempt to utilize the simple fact that the client wouldn't normally access two servers with the same session ID at the same time. This assumption allows the designer of the session manager to eliminate the potential need to lock (which will have to be a cross-instance lock) the session object.

Generation of Unique Session Identifiers

The session ID must be a unique identifier to unambiguously identify the session. Depending on the type of application, there could be different requirements for the session ID.

For a simple case of local nonpersistent sessions, this ID can be a number that is incremented every time the session is accessed, plus some time stamp to ensure the uniqueness across restarts. For multiple instances this ID must also include unique host information.

The above technique or a similar one, however, doesn't ensure the operation within an environment where security is a concern. The session ID, generated using one of the above simple approaches, could be easily guessed by a malicious client, thus allowing it to impersonate itself. The default session ID generator of iPlanet Web Server (iWS) produces cryptographically strong IDs. This, however, can bear some performance penalty, and in situations where security isn't an issue, the developer of the session manager can provide his or her own session ID generator.

Collaboration with Other Products

It's likely that the proposed Web application can be so complex that one specific set of features won't be able to satisfy all the requirements. For example, you may wish to have a lightweight, nonpersistent way of managing sessions for some part of your application, and a really elaborate, reliable session management facility for some other very critical, but not as frequently accessed, part of the application. It may be desirable to run some applications within a simple Servlet container that's built into the Web server to achieve high performance and low latencies. Some other pieces of the application may need to live within the Servlet container, which is part of the dedicated application server. It may be desirable to share the same session object with these different parts of the Web application.

Pluggable Session Management

In light of the previous discussion regarding the often contradictory needs of the server-side session container (Servlet container), it becomes apparent that there's no "one size fits all." It's essential to have some degree of pluggability in the Servlet container's session management.

During the design cycle of the iWS series, we came to the above conclusion that there could be different approaches and requirements in the way you handle session management, even within the same logical

Adrenalin new

Visualize

application. We thought it would be natural to make the session management facility pluggable so the developer could apply different session management techniques to various applications. iPlanet Web server goes even further by extending the granularity of pluggable session management to every Servlet context.

Implementing a pluggable session manager for iWS is very simple. The session manager implementer needs to extend the `com.netscape.server.http.session.NSHttpSessionManager` class. The session manager will be responsible for creating and managing session objects. Every session object the session manager creates needs to implement the standard `HttpSession` interface. Internally, however, it's up to the writer of the session manager to define the relationship between `HttpSession` objects and the session manager. Listing 3 contains the definition of the abstract class that every session manager in iWS is an instance of.

Registering your new session manager is easy – simply add a few lines to a configuration file called `contexts.properties`:

```
context.global.sessionmgr=com.Foo.Bar.mySessionManager
context.global.sessionmgr.initArgs=SomeInternalParameter=2000, AnotherInternalParameter=1000
```

iWS4.1 can potentially instantiate multiple session managers within the same server process (one per each Servlet context). The developer of the session manager has to make sure that the code is thread-safe and allows for multiple instantiations.

What's Next?

We've all witnessed the remarkable growth of the World Wide Web from a publishing medium to a complex virtual world that mirrors virtually every activity in the real world. Upcoming major advancements in the bandwidth that's available to consumers and businesses are expected to bring many more exciting features to the Web and, in turn, dramatically increase the performance and scalability requirements for Web-based applications. Inefficiencies of today's Web that are already visible will become a major challenge for services and software businesses.

Let's illustrate this with the down-to-earth example of a grocery store operation. Normally a customer in a grocery store would pick up a shopping cart, roll it through the store getting desired items, then proceed to the counter. Now let's imagine that the grocery store assigns a representative to every shopping cart. This model won't scale at all, but that's the model currently employed by a majority of e-commerce sites. By maintaining state/session information, servers are effectively rolling the clients' shopping carts, thus causing major scalability bottlenecks. This problem can be solved on both ends of the link only by providing smarter clients that understand the boundaries of the Web application, providing facilities to serialize and retrieve arbitrary structured data to and from the client and the servers, and understanding how to talk to these next-generation clients as well as older browsers.

The biggest challenge for the Web development community will be to agree on an acceptable set of standards that are going to be implemented by leading Web clients. We're hopeful and optimistic that the Mozilla open-source browser project supported by Netscape (a subsidiary of America Online) will play an important role in advancing these kinds of new technologies. ☪

AUTHOR BIOS

Ruslan Belkin, lead engineer and architect of Java Servlet and JSP support in iPlanet Web Server 4.014.1 (formerly known as Netscape Enterprise Server), has over 10 years of experience in the industry and is a member of the Servlet API expert group. Ruslan has worked on Java, CORBA, distributed objects, component models, and scripting languages, with special focus on high performance implementations of standards.

Viswanath Ramachandran is a researcher working on JSP and JavaScript support in the iPlanet Web server group. A member of the JSP expert group, he has a PhD in computer science from Brown University in the field of programming languages.

ruslan@netscape.com

vishy@netscape.com

Listing 1: State maintenance using a simple cookie

```
public class CookieServlet extends HttpServlet
{
    public void doGet (HttpServletRequest req,
        HttpServletResponse res)
        throws ServletException, IOException
    {
        Integer nof = new Integer(0);
        Cookie cookies[] = req.getCookies();

        for (int i = 0; i < cookies.length; i++)
            if (cookies[i].getName().equals("CounterCookie"))
            {
                String nofS = cookies[i].getValue();
                try {
                    nof = Integer.valueOf (nofS);
                }
                catch (Exception nfe) {}
                break;
            }
        nof = new Integer (nof.intValue()+1);
        Cookie c = new Cookie ("CounterCookie", nof.toString());
        res.addCookie(c);
        // ...
        out.println("You have visited this page" + nof + " times");
        // ...
    }
}
```

Listing 2: State maintenance using a servlet server-side session

```
public class SessionServlet extends HttpServlet
{
    public void doGet (HttpServletRequest req,
        HttpServletResponse res)
        throws ServletException, IOException
    {
        // this will create the session if it doesn't exist
        HttpSession session = req.getSession (true);

        PrintWriter out = res.getWriter();

        out.println("<HEAD><TITLE> " + "SessionServlet Output " +
            "</TITLE></HEAD><BODY>");
        out.println("<h1> SessionServlet Output </h1>");

        Integer ival =
            (Integer) session.getValue("sessiontest.counter");
        if (ival==null)
            ival = new Integer(1);
        else
            ival = new Integer(ival.intValue() + 1);

        session.putValue("sessiontest.counter", ival);
        out.println("You have hit this page <b> +
            ival + "</b> times.<p>");
        // Session ID encoded in the response URL
        out.println("<a href=\"\" + res.encodeUrl (myUrl) +
            \"\">Click here </a> if your browser doesn't support cookies");

        out.println("</BODY>");
    }
}
```

Listing 3: Base Class for all pluggable session managers

```
/*
 * Every session manager supported by iWS must extend this
 * class
 */

package com.netscape.server.http.session;

...

public abstract class NSHttpSessionManager {
```

```
/**
 * Initializes session manager. This method will only be
 * called once
 *
 * @param props A collection of init parameters passed by
 * webserver
 */
    public abstract void init (Properties props);

/**
 * Creates a session with the specified session id
 *
 * @param id      session ID
 * @return       newly created session
 */
    public abstract HttpSession createSession(String id);

/**
 * Delete a specified session
 *
 * @param session
 */
    public abstract void deleteSession(HttpSession session);

/**
 * Request an existing session with the specified session id
 *
 * @param id      session ID
 */
    public abstract HttpSession getSession(String id);

/**
 * Get default session expiration timeout in seconds
 */
    public abstract int getDefaultTimeOut();

/**
 * a method which will be called periodically to let the
 * session manager expire old sessions
 */
    public abstract void reaper ();

/**
 * Update method will be called by the server at the end
 * of every request to give
 * the session manager an opportunity to store the session
 * into reliable storage if needed
 *
 * @param session      The session object
 */
    public void update (HttpSession session); // does nothing by
    default

/**
 * Generate random session ID
 *
 * @return             random unique string which represents random
 * ID
 */
    public String generateSID ();
    // generates cryptographically-strong ID by default

    public int getMaxSession(); //
    returns 0 by default
    public int getSessionCount(); //
    returns 0 by default
}
```



CORBA's New Interoperable Naming Service

WRITTEN BY
JON SIEGEL



CORBA's new Interoperable Naming Service (INS) introduces three features that combine to help manage your computing environment and integrate it better with the Internet and your corporate intranet as well. These features:

- Define URL-like representations for the CORBA object reference, allowing your ORB to access publicly available CORBA-based services or an object that you know the name of at an available remote site that runs a CORBA 2.4 or later ORB (the CORBA equivalent of the Web's familiar format, www.omg.org).
- Let you configure your client ORB from any vendor in a standard way so as to initialize to the root-naming context you want so that, for example, all the ORBs in your company or department can see the same naming hierarchy. In fact, these extensions to runtime configurability go beyond the Naming Service, as we'll see shortly.
- Define a standard syntax for the compound name of a CORBA object so you and the CORBA 2.4 clients you exchange e-mail with will write and read the same name in the same form, and your computers will know what to do with it.

The INS was adopted by OMG in March 1999 and finished its first maintenance revision in June 2000. For the past few years, OMG has not given newly adopted specifications a release number, or issued them as a formal document, until they've been through their first maintenance revision. This makes it clear to implementers that these new specifications, although adopted, may undergo more change than specifications a year or more old, enough to be implemented as products. (Unfortunately, it also leads to some confusion about what's an adopted OMG specification and what's not.) INS finished its first maintenance revision which was released as part of CORBA 2.4, issued by the OMG in October 2000. This release included CORBA Messaging and updated the Object Transaction Service and CORBA Core. CORBA 2.4 is an interim

release; original plans called for INS and Messaging to release as part of CORBA 3.0, now anticipated in early 2001.

CORBA Object URLs: corbaloc

One thing that's great about the World Wide Web is how you can guess the Web address of a place from its company name, type it in the address window of your Web browser, and (most of the time!) get to its Web page right away. Here's a good example – OMG. You begin by guessing that our domain name is our initials, OMG, and you know we're a not-for-profit organization so our extension should be .org. You put these together and guess (correctly) that our URL is www.omg.org.

To make this work for CORBA, almost all we need to do is put a CORBA server on a well-known port of a machine with a domain name and address. What else do we need? Web servers hold a lot of different pages and serve them up one at a time, identifying them by a page or file name ("gettingstarted/overview.htm" or whatever) following the domain name and separated from it by a slash. Analogously, CORBA servers serve a lot of different objects, so we need to put an object key or object name at the end of our object URL. The object key or object name identifies the service or exact object we want a reference for.

`corbaloc` is the keyword for the CORBA URL format that translates to the object reference of an object instance, typically providing a service, at a location. Here's an example:

```
corbaloc::MyGiganticEnterprise.com/NameService
```

Taking advantage of defaults (we'll get complicated later), this URL starts with the identifying string `corbaloc::`,

followed by a domain name (here, `MyGiganticEnterprise.com`) or an IP address, a forward slash, and an object key, which in this case is `NameService`.

The Object Key

The object key is the part of the IOR that the ORB uses to identify the exact instance that is the target of an invocation. Until now – that is, in ORB-generated object references – CORBA had standardized neither the form nor the content of any object key: ORB-generated object keys are only useful to, and used by, the ORB that generated them, so there was no reason to standardize their format or content. The interoperable naming specification defines URL-like object references with object keys that humans can read.

The object key doesn't have to be a single word; it can be any string that the server responds to. Everything that follows the first slash after the domain name or IP address is the object key. For example, in:

```
corbaloc::YourDomain.org/dev/sandbox/NotificationService
```

the object key is `dev/sandbox/NotificationService`.

There's no official equivalent of a "home page" or `index.htm` for `corbaloc` invocations. `NameService`, the default object key for `corbaname` invocations, is as close as the specification comes. If you're setting up a `corbaloc` server, we suggest you use `NameService` as your object key (or one of your object keys, since nothing prevents a server from responding to a number of keys). And when you browse the Web for CORBA servers using a CORBA 2.4 client browser, we suggest you start with `NameService` as the key to the object type you search for first, at least if it's close to what you're looking for...

Standard format is key

corbaloc, Full Form, iiop Protocol

Let's take a look at the full form of the corbaloc URL, specifying every field instead of using defaults:

```
corbaloc:iiop:1.2@MyGiganticEnterprise.com:2809/pub/NameService
```

Following the URL key string corbaloc: with its single colon comes the protocol specifier. iiop: is the default protocol; you can either spell it out followed by a colon or just put the second colon as we did in the previous section. There will be URL formats for many protocols, but right now there are only two: iiop, as we've seen, and rir for Resolve Initial References. Details about the rir form are given below.

Addressing information goes between the colon that ends the protocol specifier and the slash that starts the object key. The address format is protocol specific, of course, since each protocol defines the information set that it needs to find a target. For iiop, which runs over TCP/IP, the address field includes an optional IIOP version identifier followed by @, the host specifier as either a domain name or an IP address, and (optionally) a port number preceded by a colon. The default port, registered at the IANA (Internet Association Naming Authority), is 2809.

corbaloc, rir Protocol

In addition to iiop, the specification defines an rir protocol. (rir refers to resolve_initial_references.) The rir protocol gives you access to your ORB's configured initial services through a URL. For example, passing:

```
corbaloc:rir:/TradingService
```

to string_to_object returns the same object reference as passing TradingService to resolve_initial_references. The object key in an invocation of corbaloc:rir can be any of the keys defined for resolve_initial_references; if you leave out the object key, the system inserts NameService.

When we get to corbaname, we'll use the rir capability to find our ORB's default-configured NameService and resolve a name against it in a single call.

Coding and Using corbaloc

How do you code this into your client, and how does it work? The URL is a string, and every ORB already has a standard interface that turns a string into an object reference – it's string_to_object. The INS expands the capability of this function. When you pass a corbaloc object reference string to

string_to_object in a CORBA 2.4 and later ORB, it will resolve the domain name into an address, form it into an invocable object reference, and hand you back a session reference for it. You'll have to narrow the session reference to the interface type it's supposed to be – NameService, TraderService, or whatever your object key corresponded to.

If you're the server programmer and these invocations are coming at you, there are (at least) two ways to respond to them: a normal CORBA name or trader object instance will recognize the incoming request and do the right thing, so you can just run yours on your main host IP address at the registered iiop port 2809. Or if you don't want to run your server on your gateway machine, you can run a lightweight software agent that listens there and replies to Request or LocateRequest messages with LOCATION_FORWARD replies. Any standard CORBA client will reissue its request to the location you've forwarded to, allowing you to shift this load away from your gateway host to a more suitable server. Your agent will have to run on a CORBA 2.4 ORB and recognize these newly defined object keys; if you don't use an agent, you'll have to use a CORBA 2.4 ORB.

URLs for Objects by Name: corbaname

corbaname builds on corbaloc to let you resolve a name in a remote name service. Here's a sample corbaname URL:

```
corbaname::MyGiganticEnterprise.com/Pub/Catalogs#Year2000/Menswear/Outdoors.obj
```

Up to the pound sign (#), it's a corbaloc URL except that the identifying string is now corbaname. After the #, it's an object name in the newly standardized OMG format that we'll present in our final section.

When you pass this to string_to_object, your ORB resolves the corbaloc part to a NamingContext object instance in the host's Naming Service. Your ORB then invokes resolve on the NamingContext instance to resolve the name part of the URL.

For our example the client ORB would attempt to resolve the name Year2000/Menswear/Outdoors.obj on the NamingContext instance Pub/Catalogs on the CORBA server at MyGiganticEnterprise.com that was listening on port 2809.

If you leave out the starting context, the object key (which is what it is at this spot in the corbaname string) defaults to NameService.

You can use the rir protocol with corbaname. If you leave out the object key, it defaults to NameService. Thus, invoking string_to_object with:

```
corbaname:rir:#pub/Catalog/Fall2001/Shirts.obj
```

would find the default-configured Name Service and resolve the object named pub/Catalog/Fall2001/Shirts.obj, all in a single call.

Configuring Initial Services and References

You can do many things by setting the root-naming context at client startup. For example, by pointing all users' ORBs to the same root, you can unify the namespace across your enterprise's entire computing structure, allowing your users to exchange object names by e-mail and have them work everywhere. Developers, on the other hand, could initialize to either their own individual root-naming context or a workgroup context, depending on how their work was structured. Some will need one root context for one project and another for a second. Administrators will also need control and flexibility in this setting.

OMG's original specification for client-side initialization half-defined the services list_initial_services and resolve_initial_references, covering how to pull 'em out pretty well, but leaving the stuffing 'em in part to the ORB vendors' discretion. As a result, some ORBs came with their own Naming Services initialized to them and left no way for programmers or users to change this setting, while others didn't initialize to anything unless you set it in an environment file or command-line argument at startup.

The INS RFP required the specification to cover the setting of initial object references for the NameService, and while they were at it, the authors of the specification extended it to cover all initial services. They provided three hierarchical settings:

- Administrative setting
- ORBInitRef
- ORBDefaultInitRef

Once we've covered them, I'll tell how they interact as your ORB starts up.

Administrative Setting

The specification requires, in about these words, that an ORB be administrably configurable to return an arbitrary object reference for nonlocality-constrained objects – that is, the standard services that reside outside the ORB.

(After all, it wouldn't make a lot of sense to let a system administrator redefine the object reference of the RootPOA!)

ORBInitRef

You can set the object reference for a service at startup by passing to the ORB an argument pair of the form:

```
-ORBInitRef <Object ID>=<Object URL>
```

<ObjectID> may be either a string from the list of initial services or a new ObjectID; that is, you may define and initialize a new initial service ID that wasn't there when the ORB was written.

<ObjectURL> can be any of the URL schemes supported by CORBA::ORB::string_to_object, including the ones we've just discussed. For example,

```
-ORBInitRef NameService= IOR:
00230021AB08FA123...
```

configures the NameService using a stringified CORBA IOR, and:

```
-ORBInitRef NotificationService=
corbaloc::TelecomSvcCo
/NotificationService
```

configures the Notification Service using a corbaloc-format object reference.

ORBDefaultInitRef

ORBDefaultInitRef lets you define the default host for your services to be one location or a cascading series of locations represented by a URL-format object reference. You provide all of the URL except the slash (/) character and object key; your ORB appends the slash and the appropriate key string and feeds the resulting string to string_to_object. For example, if you input:

```
-ORBDefaultInitRef corbaloc::My Ser
viceHome.com
```

at ORB startup, and then called resolve_initial_references ("NotificationService"), your ORB would add a slash and object key to generate the now-complete URL:

```
corbaloc::MyServiceHome.com/ Notifi
cationService
```

and pass it to string_to_object to obtain the session reference for the service, which it would then return to you. As you continued to resolve_initial_references, each new request would generate a URL with the appropriate key string, and your ORB would resolve just as it did the Notification Service in this example.

Resolution Order

These three methods are resolved in a strict order, which depends on how your ORB is configured. The default order is:

- First, attempt to resolve using -ORB-InitRef. If you didn't specify an ORBInitRef for this service at initialization, or if the resolution attempt fails,
- Attempt to resolve using -ORBDefaultInitRef. If you didn't specify an ORB-DefaultInitRef, or if the resolution attempt fails,
- Attempt to resolve using the administratively configured settings.

In some cases you don't want an ORB to use the default setting to divert certain services. The specification suggests, for example, that an ORB that uses a proprietary Implementation Repository may not want it to be diverted to use one from another vendor or one that doesn't work in the way it expects or requires. To prevent this, an ORB is allowed (but not required) to ignore the ORBDefaultInitRef argument when it resolves services that don't have a well-known service name accepted by resolve_initial_references. The ORB is not, however, allowed to ignore any setting specified by ORBInitRef – only ORBDefaultInitRef.

String Form of CORBA Names

The original CORBA name service specified how names were stored inside your program (as a struct) but provided no standard format to print them out. That meant you couldn't print out the compound name of a CORBA object and send it to a friend or co-worker and expect him or her to use the literal string unless you both had the same ORB and used the same code. (The name could be parsed into pieces and each piece entered individually if your friend's GUI allowed it, but this isn't a friendly way to deal with names!) Another reason to fix this deficiency: corbaname requires a standard representation for the string form of a compound name. The revision of the name service fixes this by specifying a string representation for CORBA names, which it refers to as *stringified names*. (Careful: not the same as a stringified object reference!)

The bottom line is that CORBA object names are UNIX-format directory and file names applied to objects. If you're more used to DOS directory and file names, change the backward slashes to forward slashes – everything else is the same. If you're used to Windows names, change the backward slashes to

forward slashes and don't use spaces. There are some other differences but they're subtle and we won't go into them here.

At each level CORBA object names – whether for another naming context or an object – already consist of an identifier or ID (the main part of the name) and a kind (the minor part). In the newly defined standard name string, these two parts are separated by a dot, and the different naming contexts are separated by slashes. (In the original Naming Service these were represented as a struct in your program, and no representational syntax was defined.) Just in case you wanted to put dots and slashes into your object name, OMG defines escape characters.

Two stringified names are equal if every identifier and every kind in one equals the corresponding name and kind in the other. Comparisons are case sensitive. If the two strings are equal, the names are equal.

Converting Among Object Name Formats

A new interface, NamingContextExt, inherits from the NamingContext interface and extends it with operations that convert among the forms of object names: to_string and to_name do just what you expect. resolve_str is a combination of to_name followed by resolve. And to_url combines a corbaloc address/object key combination and a stringified name into a corbaname string.

Summing Up

That's a summary of the Interoperable Naming Service. Although I've left out a few details, the major features are covered. It goes a long way toward integrating CORBA into the Internet and corporate intranet world, where it already plays a significant role, so I'm pleased to see it released with CORBA 2.4. In future columns I'll cover the Messaging Service and the changes to CORBA core that were also part of the CORBA 2.4 release.

• • •

The version of the INS current as this column was written may be downloaded from www.omg.org/cgi-bin/doc?orbos/98-10-11, and www.omg.org/cgi-bin/doc?ptc/1999-12-02, -12-03, and -12-04.

This article was adapted from material in my latest book, *CORBA 3 Fundamentals and Programming*, and a forthcoming book, *Quick CORBA 3* (Wiley). ☛

siegel@omg.org

AUTHOR BIO

Jon Siegel, the Object Management Group's director of technology transfer, also writes articles and presents tutorials and seminars about CORBA. His new book, *CORBA 3 Fundamentals and Programming*, has just been published by John Wiley and Sons.

WRITTEN BY
IVAN KISELEV

All major and minor application server vendors heavily advertise the connection pooling functionality of their respective offerings. In this article I examine what's involved in developing resource pooling features from the perspective of a Java developer. I feel the subject is both greatly overhyped and underrepresented in technical literature.

The Problem

First, I'll try to describe the problem in very general terms. I'll rely more on practical considerations rather than scientific completeness:

- A set of resources has one common characteristic: there's a limited supply. Usually the supply limitation isn't related to the scarcity of the resource, but to the difficulty of obtaining it. For example, it's possible to open a database connection for each query, but it's a time-consuming operation.
- It's assumed that these resources will be needed by different parts of an application that will "compete" for them according to some application logic.
- Each individual resource can be used an unlimited number of times.

PRODUCE

A RESOURCE

POOL FOR

Translating plain English into Java, it looks like a collection of resources that:

- Contain a pool of resources to be managed
- Supply the needed resource on demand
- Deny the resource if none is available
- Exercise some policy on resource availability including queuing, waiting, and so on

Java's threading model lends itself nicely to shared access to resources. In this article I assume that resources are shared among concurrent threads, not among the operating system's processes. The implementation described below can be easily adapted for a multiprocess situation by developing an interprocess communication layer on top of it.

Resource Pooling In Java

Generic Implementation

The generic implementation is provided in the form of the class Pool (see Listing 1).

Theoretically speaking, the Pool class can be a Stack; for example, Pool can extend Stack with the Pool.pop(long time) method, override Stack.pop() and used (semantically) in place of tryPop(). Nevertheless, I've chosen delegation over inheritance and the reasons are:

- The Stack.pop(), and Pool.pop() methods have different semantics. The former doesn't really "try" to pop it; it's quite unconditional in its attempt to pop the object and will throw an exception if unable to do so.
- The other methods of the generic Stack class will violate the integrity of the pool. For example, Stack.peek() will deliver a potentially shared object to anybody who asks; Vector.clear(), inherited from the superclass, will destroy a stack-based Pool altogether. You can also "mute" all of Stack's methods that Pool doesn't need, but it will confuse the class's users even further.

Let's consider how the Pool class can be used. First, an application must create a Pool object, usually a static one. Second, the Pool must be populated with the resources to be managed. After that the Pool is ready to use (see Listing 2).

HEAVY LIFTING

In Listing 2 the method `usageExample()` uses a static instance of the `Pool` class (named `staticPool`) created and initialized elsewhere in the class. The type “Resource” represents any particular type of resource that the application will need. The resource is obtained from the pool via the `pop()` method that accepts a waiting time parameter (in seconds) (e.g., the `Pool` class will try for 10 seconds to obtain the resource, if it’s not currently available). During that time the pool will check 10 times if some other thread put any of the resources back and will return it if so. If none of the resources become available, the pool will return null and it’s up to the application to decide how to proceed further; the example in Listing 2 just gives up and throws an exception.

An important aspect of using pools is that a resource obtained from a pool must be returned after the application is done using it. This is guaranteed by the “finally” clause in the example: if the resource was successfully obtained (not null), then it will be returned (call to the `Pool.push()` method). Obviously, if anything prevents the return of the resource, it creates a resource leak that will eventually deplete the pool.

The synchronization of the `Pool`’s methods deserves special attention. Since `java.util.Stack` class is used in implementing the `Pool`, there’s a certain sense of security as far as multithreading is concerned. The `Stack` class is, in turn, implemented as a `java.util.Vector`, which is properly synchronized. Unfortunately, this sense of security proves to be a false one and for the following reason: method `Pool.tryPop()` prevents the possibility of popping the empty stack (with related complications in the form of a nasty runtime `EmptyStackException`) by checking it via `Stack.empty()`. The problem arises if the execution path veers off the current thread after the `Stack.empty()` call and some other thread gets the last resource in the pool, then the current thread does pop an empty stack! It’s hard to estimate how often it will happen, but it’ll be too late to think about it when exceptions start flying out of your complete and attractively shrink-wrapped product. So we have to synchronize the `Pool.tryPop()` method; however, the rest of `Pool`’s methods don’t need any additional synchronization. `Pool.push()` is synchronized via its superclass `Vector` and `Pool.pop(long time)` does all its magic via `Pool.tryPop()`.

Exercise 1

The `pop()` method uses a very simple waiting algorithm – it tries to get the resource 10 times per waiting period. The overall performance of the system that uses resource pools can be significantly improved if a more adaptive waiting algorithm is utilized. One such algorithm I’m aware of can be described as follows: start with a very short waiting time (say, 1/100 of the total wait) and double it with each iteration until either the waiting limit is exhausted or a resource is obtained. The effect is that if no resource is available immediately, the pool will try more frequently in the beginning of the wait and less so toward the end; it can be called a “vanishing hope” method. For a system that operates near its performance capacity, it results in better response time during occasional overloads. The implementation of improved waiting schemes is left as an exercise to the reader.

Database Connections

The resource pooling is most frequently used for database connections. Naturally, database connections are presumed to be JDBC connections for the purposes of this discussion. Several specific database issues warrant special consideration with regard to resource pooling:

- A database connection can and will be broken, in my experience, during the application instance’s life span. Some drivers (could be all of them) may not restore connections if their respective database servers are restarted.
- It’s impractical to create all possible database connections during an application’s start-up. It takes forever and all these connections may not be needed.

These issues lead to the following requirements for database connection pooling:

- Connections should be periodically checked for validity.
- Connections should be dynamically allocated as needed.

Listing 3 presents an implementation based on a generic `Pool` class that was discussed in the previous section. *Note:* A lot of details were omitted for the sake of clarity, namely, initialization, synchronization, and proper exception handling.

The `DbConnectionsPool` class extends the generic `Pool` with the following functionality:

- It can create new database connections. All parameters that are needed to do so (driver, user name and password, maximum number of connections allowed, etc.) are passed to the class constructor and later used by the `createConnection()` method.
- Its constructor starts a thread that periodically checks connections for validity and destroys bad ones.
- The `pop()` method creates new connections (up to the specified maximum number) when the pool is depleted.
- `isGood()` method tries to send a “null-operation” to the database just to make sure it can go to the server and back. Since all the connections to be tested aren’t used by any pending transactions, the statement “rollback work” should do nothing (check with your particular database’s documentation).

Exercise 2

The `run()` method removes a connection only when it goes bad. It might be very useful to create an algorithm that will destroy connections according to their usage patterns. For example, the winning strategy might be to create new connections when needed and remove old ones when demand for them is low. It’ll reduce memory requirements of the application and may ease up the load on the database server. An implementation of such an algorithm is offered as an exercise. *Hint:* Compare `currentConnections` counter with the number of connections in the pool while checking for validity. If these two numbers are close, then connections are underutilized.

CORBA Connections

The CORBA world offers its own set of complications as far as resource pooling is concerned. The good news is that CORBA connections are automatically rebound (that’s CORBA-speak for “reconnected”) in case of failure (at least, the implementation I use – Inprise’s `VisiBroker` – does this). The bad news is that the CORBA connection pool implementation has to deal with the fact that CORBA connections, unlike JDBC ones, are not created equal. A typical application usually connects to several different CORBA objects that are distinguished by their type and name.

It’s quite possible to create separate connection pools for each object type and name combination, but it leads to code bloat and, generally, reduces the reliability and maintainability of an application. Let’s see how the situation can be remedied.

Consider the `CORBAConnectionsPool` presented in Listing 4 (again, many implementation details are omitted). The basic idea is simple: create a separate pool for each object name and automatically open new connections using the same strategy that was used with the `DbConnectionsPool` class. More details on this approach:

- We distinguish CORBA objects by specifying their respective names and IDs as parameters in `pop(...)` methods.
- The object ID can be omitted if an object inherits from a `GenericObject` CORBA interface. The `CORBAConnectionsPool` class is aware of this interface and “knows” how to get its object ID, creating a valid ID to bind to any of its subclasses. The right reference can be obtained by the actual application by narrowing down (CORBA’s equivalent of type casting) the reference to the `GenericObject` retrieved from the pool.

Unfortunately, as with any compromise, these implementation details force the user to exercise some caution regarding what can be safely stored in this pool. The following convention must be observed: an object name must uniquely identify an interface for the purposes of the `CORBAConnectionsPool` class. For example, if there are two interfaces, `Laser` and `Bubble`, that extend the `Device` interface, and both CORBA services that implement `Laser` and `Bubble` are named the same (say, “Printer,” or not named at all), the code in Listing 5 would lead to storing an

ambiguous reference in the CORBAConnectionsPool that was caused by the polymorphism. The same will happen if we try to manage multiple objects with the same name and completely unrelated types, but for a different reason: the CORBAConnectionsPool class doesn't keep track of object IDs along with object names, which leads directly to Exercise 3.

Exercise 3

Since the above-described implementation distinguishes CORBA objects only by their names, develop a CORBA connection pool that will also account for object IDs.

The overall usage pattern for CORBAConnectionsPool follows the example presented for a generic Pool in Listing 2 with changes dictated by CORBAConnectionsPool method signatures. Again, as always, it's extremely important to guarantee the return of the used connection back to the pool.

The most compelling case for using CORBAConnectionsPool despite its object ID indifference is when an application has to deal with a large number of objects of the same interface, for example, printer services on a network. It's also quite convenient for the CORBA services that are part of a larger system to implement some common generic interfaces, such as GenericObject, in the example above. This interface can be used to implement common interface features that are related to some application group (e.g., for life-cycle management).

Conclusion

As you can see, it's a relatively straightforward task to develop a resource pooling solution in Java, thanks to built-in multithreading. Unfortunately, a developer has to take care of a lot of implementation details (careful synchronization is one example) to produce a resource pool for production-level heavy lifting. Even more unfortunate, domain-specific pools require drastic changes to the interface compared to

generic ones. Java development of resource pooling can be taken a bit further by exploring the possibilities of developing a general framework that includes these features:

- **A flexible mechanism for defining resource availability policies that vary depending on the application:** For example, a database has different performance characteristics than a Web server and, therefore, these two require different resource strategies including waiting and queuing.
- **Ability to accommodate resources of various types without any resource-specific knowledge on the framework's part:** As shown above, a database connection requires different caretaking compared to a CORBA link. It should be possible to provide generic interfaces for resource creation, initialization, and upkeep regardless of the particular type.
- **"Learning" runtime behavior with regard to resource creation and clean up:** For example, it should be possible for some applications to predict their demands for a particular resource type and prepare the resource pool accordingly. For example, authentication services are loaded mostly in the morning, printers are at night, and the fax load depends on the long-distance fee schedule.

While exploring future possibilities of resource pooling, it's important to concentrate on the usability of the perspective solution: interfaces that are too general or complex may encourage developers to choose a resource-specific solution. ☺

AUTHOR BIO

Ivan Kiselev is chief architect at APP Design Group, Inc. His professional interests include applications of reusable frameworks and application servers to electronic commerce systems, development environments, and integration of scripting languages into all of the above.

ikk@appdg.com

Listing 1

```
import java.util.Stack;

public class Pool
{
    // This is a storage
    private Stack stack = new Stack();

    public Object pop(long time)
    {
        long timeLimit =
            System.currentTimeMillis()+time*1000;

        while(System.currentTimeMillis()
            < timeLimit)
        {
            Object anObject = tryPop();
            if( null != anObject )
                return anObject;

            // Sleep for time*1000/10, i.e. 1/10
            // of time allowed.
            Thread.sleep(time*100);
        }

        return null;
    }

    public synchronized Object tryPop()
    {
        if(stack.empty())
            return null;

        return stack.pop();
    }

    public void push(Object obj)
    {
        stack.push(obj);
    }

    public int size()
    {
        return stack.size();
    }
}
```

Listing 2

```
public void usageExample() throws Exception
```

```
{
    Resource res = null;
    try
    {
        res = (Resource) staticPool.pop(10);

        if( null == res )
            throw new Exception("Resource is
                unavailable.");

        // Do something with the resource
    }
    finally
    {
        if( null != res )
            staticPool.push(res);
    }
}
```

Listing 3

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

public class DbConnectionsPool
    extends Pool implements Runnable
{
    // Connections counting
    private int maxConnections;
    private int currentConnections = 0;

    // Thread that does connection checking
    private Thread checking = null;

    // Time interval (in seconds) between
    // connection checking attempts.
    private static final int
        CHECKING_INTERVAL=60;

    // JDBC-specific variables go here
    ....

    public DbConnectionsPool(
        int maxConnections, String dbDriver,
        String dbUrl, String dbUser,
        String dbPassword)
    {
        // Initialization of internal
        // variables goes here
        ....
    }
}
```

```

// Create a checking thread
checking = new Thread(this);
checking.setDaemon(true);
checking.start();
}

public Object pop(long timeOut)
{
    Connection conn = null;

    // Try to get it from the internal stack.
    conn = (Connection) tryPop();
    if( null != conn )
        return conn;

    // Try to create another connection
    // if possible.
    if( currentConnections < maxConnections )
    {
        conn = createConnection();
        if( null != conn )
        {
            currentConnections++;
            return conn;
        }
    }

    // Wait if cannot do any better
    return super.pop(timeOut);
}

private Connection createConnection()
{
    // Code to create a JDBC connection
    // goes here
    ....
}

private boolean isGood(Connection conn)
{
    boolean result = true;
    Statement stmt = null;
    try
    {
        stmt = conn.createStatement();
        if( null != stmt )
            stmt.execute("rollback work");
    }
    catch(SQLException e)
    {
        result = false;
    }
    finally
    {
        if( null != stmt )
            stmt.close();
    }

    return result;
}

public void run()
{
    Stack holding = new Stack();

    while( true )
    {
        // Wait CHECKING_INTERVAL seconds
        Thread.sleep(
            CHECKING_INTERVAL*1000);

        // Get all available connections
        // into a temporary stack
        while( true )
        {
            Connection conn = (Connection) tryPop();
            if( null == conn )
                break;

            holding.push(conn);
        }

        // Go over the stack and check connections
        while( !holding.empty() )
        {
            Connection conn = (Connection) holding.pop();
            if( isGood(conn) )
            {
                // Put it back if it's good
                push(conn);
            }
            else
            {

```

```

// Drop it if it's bad
conn.close();
currentConnections--;
}
}
}
}
}

Listing 4
import java.util.Hashtable;
import org.omg.CORBA.*;

public class CORBAConnectionsPool
{
    private Hashtable
        connectionPools = new Hashtable();
    private ORB orb = ORB.init();
    private final int maxConnections = 10;

    public org.omg.CORBA.Object pop(String id,
        String serviceName, long timeOut)
    {
        // Try can get a connection pool from
        // the hashtable.
        Pool pool =
            (Pool)connectionPools.get(
                serviceName);
        if( pool == null )
        {
            // Initialize ConnectionPool
            pool = new Pool();

            connectionPools.put(serviceName,
                pool);
        }

        // Try can get a connection from the pool
        org.omg.CORBA.Object corbaObj =
            (org.omg.CORBA.Object) pool.tryPop();
        if( corbaObj == null )
        {
            // Try to create another connection
            if( pool.size() < maxConnections )
            {
                // WARNING: this is VisiBroker-
                // specific call
                corbaObj =
                    ((com.visigenic.vbroker.ORB)orb).
                        bind(id, serviceName, null, null);

                if( corbaObj != null )
                    return corbaObj;
            }
        }

        if( corbaObj == null )
            corbaObj =
                (org.omg.CORBA.Object)pool.pop(timeOut);

        return corbaObj;
    }

    public org.omg.CORBA.Object pop(
        String serviceName, long timeOut)
    {
        String id = GenericObjectHelper.id();
        return pop(id, serviceName, timeOut);
    }

    public void push(String serviceName,
        org.omg.CORBA.Object corbaObject)
    {
        ((Pool)connectionPools.get(serviceName)).
            push(corbaObject);
    }
}

```

Listing 5

```

void ambiguousExample()
{
    CORBAConnectionsPool connPool =
        new CORBAConnectionsPool();

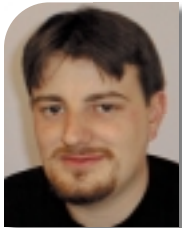
    org.omg.CORBA.Object corbaObject =
        connPool.getConnection(
            DeviceHelper.id(), "Printer", 10);

    // WRONG!!! It could refer to Bubble
    // interface.
    Laser interface =
        LaserHelper.narrow(corbaObject);
}

```



Ahoy There, Me Hearties!



WRITTEN BY
ALAN WILLIAMSON

With the fourth quarter upon us, it's good to see Christmas coming around again – more important, things are happening in the wide world. This time last year everyone was panicking over the Y2K problems they'd be experiencing and the havoc that would be created by having the world's computers come crashing down. It's all gone fairly quiet now, don't you think? Not a single report have I read anywhere on post-Y2K problems. Evidently not that newsworthy.

That said, I think there should be a follow-up on those individuals that were interviewed building nuclear-style bunkers and piling up food supplies. I'd love to know just how stupid those people feel now. The religious clowns who thought this was the big checkout must be feeling fairly let down as well. Oh well, always jobs in the dot-com world, eh?

R.I.P.

Not if you read the papers. It still makes sorry reading. Maybe I should rename this column "dot.com Obituary"? Oh, here, that reminds me, before I head off into detailing yet another dot-com that's gone pear-shaped, check out these sites:

www.startupfailures.com/
www.dotcomfailures.com/

No matter what happens, somebody always finds the silver lining. Here when the very foundation of our industry is on rocky ground, somebody has got funding for an idea that relies on failures. Some analysts would say madness – I say pure genius. I love this industry; you never know what's around the corner.

What's been hailed as a veteran of European e-tailers, boxman.com, is looking to go into voluntary liquidation. Now they're looking for a buyer, so by the time this column is published they may have gotten a reprieve. Historically, this doesn't look good. They seem to be struggling to raise additional capital to stay in business. I've come up with a theory on why these dot-coms are going to the wall, but more on that later.

Boxman.com sold music CDs as their primary business. A commodity that didn't involve a huge shipping overhead, yet they still aren't making any money after three years of trading. That troubles me. I can understand sites pushing less conventional e-items (hey, I've coined a new phrase...cutting edge this column, I tell you) struggling to break even. It can take a lot of technology cost overhead to make the site informative enough to allow a shopper to make the decision to complete the purchase or not. I read a report saying that any companies making it past Christmas will have ridden out the rough period. Interesting. Not sure if I believe it, though.

Crystal Ball

Let me give you a great way to spot whether a dot-com is doomed to fail. Now before you get too excited, it's not a proven technique, but so far the process is holding up very well. Are you ready? One word: *celebrity*. If a dot-com has a celebrity listed on its board of shareholders, be forewarned. Remember clickmanago.com? It had Joanna Lumley as an investor (before you ask, she's the one from the British export "Absolutely Fabulous"). Boxman.com, our latest going-down-the-tubes dot-com, has the Swedish rock band Roxette and Madness singer Suggs on its list of investors.

Keep an eye out for the celebrities and where they put their money. Let me know if you know of any other famous investors. An interesting and fun exercise to find out who's got who.

Forget Wireless

Everyone and their dog is jumping on the WAP revolution. "Internet on the



move!" they cry. It's nonsense. I have seen the future and it is not on the move, but at home. It's called broadband. We've just had our 2Mb line installed, and let me tell you, it's *fantastic*. The speed at which Web sites pop up is breathtaking. Napster seriously rocks! Or it did at this writing. But more important is the ability to watch streamed video. Whoa! We've got ourselves set up with a computer that just plays music videos continuously from the Net. The quality is perfect.

We were thinking about this and there are three major experiences of the Internet at the moment. First are the poor users who have to view the Internet via a dialup connection. They have to collect their e-mail as opposed to having it delivered. They have to put up with poor download speeds complete with connections that drop out. A lot of the Internet is shut off to them as they simply don't have the connection speed to view the material.

On the second level are those who are on a permanent connection but still below 256KB speed. They have the benefit of getting e-mails when they're sent. They also get to see all the utilities that require a permanent connection, such as automatic news feeds, ticker tapes, images that update by themselves, and more reliable gameplay. Generally speaking, they're a little more relaxed when viewing the Web. They don't need to worry about the online costs as they're fixed.

The third and most rewarding level has to be what's now categorized as *broadband users*. These are the ones that are sitting with a T1 connection and above. Surfing becomes a wonderful experience, opening up the world of

real-time radio and TV. In addition to this, video conferencing becomes a serious reality and the overall attitude of the Internet becomes one of a tool as opposed to a novelty.

So going back to a medium that at the moment delivers around 9.6Kbps isn't really going to ring my bell. Writing applications for this platform isn't nearly as exciting as getting your teeth into the Java Media API and producing some really cool streamable applications.

Now I'm not writing off wireless computing – that would be more than a little naïve of me. But as soon as the marketing people start giving the real picture of wireless computing as opposed to the “multimedia experience on the move” speech, the better the world will be. People's expectations have been encouraged to be completely unrealistic, and the fall is so much harder when they discover that the reality of the experience is something along the lines of the ZX Spectrum.

Cisco \$\$\$\$ Oracle

It's interesting to watch how the established companies are playing in the buyout game. Many of them are adding to their suite of products by buying the technology and integrating it. Nothing new in that, you say. But how

they're paying for it is. There's a term, *Cisco-dollars*, which refers to stock as the currency in the trade, so, generally speaking, it costs Cisco very little hard cash to make purchases. Oracle, on the other hand, pays in cold, hard cash. I read that Oracle feels their stock is worth far more than money and they want to hold on to it as long as possible. A very brazen and confident move. I like Larry's style.

Pirates

This week I caught the infamous TV movie from TNT. It's just made it out on video over here in the land of the kilted warriors. The tape I allude to is the story of Apple and Microsoft: *Pirates of Silicon Valley*. I was intrigued. As you know, I read a lot of books on the history and backgrounds of companies. I've read many different books on the same company so I have a good perspective on the truth, or at least I get to read two sides of a purported truth. So I watched the movie with much background information, expecting it to be nowhere near it.

On the whole, I was impressed. They stayed pretty much to the core story, with artistic license exercised at a minimum. One thing I did find interesting was the portrayal of Steve Jobs. He came across as a right bastard to work for. He

must have really sold early Apple employees the bigger dream for them to put up with that amount of abuse. I've read in a number of books that Steve Jobs was a hard man, but I never realized he was quite this harsh. Maybe the movie overdramatized this...I don't know. If any of the early Apple employees are reading this, *please* get in contact with me. Would love to know what it was really like.

We live in an exciting time, people, and it gives me such a buzz to know that these pioneers are still working among us. We're lucky. Our children will only know these people as old and past it, but we have the opportunity to work with these people and shape the future.

Hurrah

Remember to keep an eye on the Web site for updates: www.n-ary.com/industrywatch/.

See you next month. ☺

AUTHOR BIO

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

alan@n-ary.com

Short List p/u (Ape)

LOGGING



Third-party logging API helps trace and debug problems

WRITTEN BY
VINAY AGGARWAL

During our last project we needed a logger but didn't want to develop our own, so we looked for third-party logging APIs. We found a few and experimented; one of them, *log4j*, far outshone the others. It helped us so much in tracing and debugging our problems that we recommended it for other projects, where it also got rave reviews.

This API was developed by IBM at their Zurich research lab (www.zurich.ibm.com). Initially, IBM posted it on its alphaWorks Web site (www.alphaworks.ibm.com), but has since moved it to an open source organization, Source Forge (www.sourceforge.net). It can also be accessed at www.log4j.org.

The Need for Tracing and Logging

Two methods are widely used for debugging: you can either step through the code in a debug environment or you can add debug statements in your code, run the code, and analyze the log generated. The problem with the first method is that it can take a lot of time to step through the code, which can be done only as part of the development phase. In addition, it's difficult to get an overall view of the program execution as you only have the current state of the program at any given time. To get an overall view you have to record the values of the data as you step through the code, which is equivalent to adding debug and trace statements. The advantage of adding debug statements is that this mechanism is persistent. You can see the log anytime you wish (even after the system has been put into production), instead of stepping through the program every time.

Developers often use `System.out.println` statements to get desired values. Though this provides some quick help to developers, it's not a good solution in the long term. These statements aren't flexible (i.e., time, thread information is generally not logged, and the information to be logged is hard-coded).

If you want to change the format, you have to edit the entire code, and it can't be used in environments where console output isn't available (e.g., server-side components, EJBs, servlets, JSPs). If performance is an issue, these statements have to be removed (or commented out) before the system goes into production. If there's a problem in production, the debug statements need to be inserted again, code needs to be compiled, and the application needs to be restarted. This obviously has its drawbacks.

Using a logging API solves these problems. Often these APIs are custom-made (leading to increased project costs), less flexible, unoptimized, and sometimes even unstable. I've used two such APIs and haven't been happy with them.

The Basic Stuff

The `log4j` API allows you to log text messages as well as exceptions. (It logs the passed message and then exception stack trace.) You can put logging statements in catch blocks, and the log generated can be very helpful in debugging.

Let's start with the concept of priorities (severities). Each logging message has to be put with a priority. Generally used priorities are `DEBUG`, `INFO`, `WARN`, and `ERROR` (four more priorities – `NOTICE`, `CRIT`, `ALERT`, `EMERG` – exist but they're discouraged as they get confusing and aren't really required). You can configure the `log4j` package so it only logs messages above a given priority.

Suppose you've configured the package with `WARN` priority. All messages with priority `WARN` and `ERROR` will be logged, and all messages with priority `INFO` and `DEBUG` will be ignored. During development you can configure it to log `DEBUG` messages, and during production all you have to do is configure it to log all messages with priority `INFO` or higher. These debug messages won't be logged. This obviously gives cleaner and less verbose logs and saves time in production.

Three functions are available to add debug messages:

- `debug(String message)`
- `debug(Object message)`
- `debug(String message, Throwable t)`

These functions will log a message with debug priority. Similar convenience functions exist for `INFO`, `WARN`, and `ERROR` priorities. For other priorities (which are used less often) such functions don't exist. You can always use the following general functions to log a message:

```
log(Priority priority, String message)
log(Priority priority, String message, Throwable t)
```

Categories

Now that we're done with the simple concepts, let's talk about slightly more complex things like categories. The logging API supports multiple logs at the same time. This means you can log different messages into separate files, each with its own format, at the same time. Defining categories does this. Before any log statement you have to get the `Category` object. This is done by specifying the category you want to log to (e.g., `employee`, `finance`). I'll illustrate the usage with an example:

Get the `Category` object as follows:

```
Category cat = Category.getInstance(getClass().getName())
```

This returns a `Category` object that has all the methods used to log the messages.

The recommended use is to organize your code according to different modules and put them in separate packages (I guess most of us do this anyway). For instance, your employee-related classes would go in a package called `com.mycompany.employee` while your finance-related classes would go into `com.mycompany.finance` package. Now all the log statements in the code in `employee` packages will log into the category `com.mycompany.employee`, and similarly, all code in the `finance` package will log using category `com.mycompany.finance`.

Though you can configure the categories using various configuration functions in the API, a better option is to use a configuration file. Your settings can be specified in a property file and `log4j` can load it to configure the categories (see Listing 1). The first three lines show three categories: `com.mycompany.employee`, `com.mycompany.finance`, and `ROOT`. If you try to get a `Category` object for a nonexistent category (such as `com.mycompany` or `foo.bar`), it will be created and returned. This new category will inherit its properties from the hierarchy.

The Hierarchy Story

Does this mean you have to configure a different category for each package? No. A very powerful feature of the `log4j` is category hierarchy (much like a class hierarchy). Just define a category – `com.mycompany` – and all the logging messages logged with category names – `com.mycompany.employee`, `com.mycompany.finance` – will be logged to category `com.mycompany`. Now suppose you find some bug in the `employee` package. All you need to do is define another category, `com.mycompany.employee`, that logs to a different file with severity as `DEBUG`. All the debug output would go to a different file, which can be deleted after debugging.

The logging categories are hierarchical just like packages in the Java language. This means that if you have a category named `com.mycompany` and another one, `com.mycompany.employee`, the category `com.mycompany.employee` will inherit the properties of `com.mycompany` (including appenders), unless it's specifically configured to override inherited properties.

Does this mean that all the messages you log under `com.mycompany.employee` will also be logged under `com.mycompany`? Three rules determine this. They're well documented, with examples, in the `Log4j` documentation. I can't reproduce all of it here, but here are the rules:

1. The chained priority for a given category `C` is equal to the first nonnull priority in the category hierarchy, starting at `C`.
2. A log statement of priority `p` in a category with chained priority `q` will be printed if `p >= q`.
3. The output of a log statement of category `C` will go to all the appenders in `C` and its parents. However, if a parent category of `C` has the additivity flag set to false, then appenders of that parent aren't used.

If you don't want to log `com.mycompany.employee` messages to `com.mycompany` category, use the additivity setting. Setting additivity to false for any category will cause it to stop sending its messages to its parent category. Look at line 4 of Listing 1; this tells us that for the category `com.mycompany.finance`, you shouldn't send the log messages to the `ROOT` category. But as nothing is mentioned for the category `com.mycompany.employee`, the messages from this category will be logged in the `ROOT` category as well. Listing 2 contains a sample code that uses Listing 1 to configure categories. You might need to experiment with it to get more comfortable with the package.

Appenders

For each category you can specify an appender – an object that will finally send messages to the required destination. There's a `FileAppender`, which will write the logs to a file, and the `RollingFileAppender`, which will roll over files beyond the specified size. It will rename the old file and start again in a fresh file. A `SocketAppender`, which will dump the logs to a socket, can be useful if you're building a distributed logger. An `NTEventLogAppender` will write the logs to the NT Event Log System.

Note that NTEventLogAppender works only on Windows NT. Similarly, there's a SyslogAppender to log messages to a remote syslog daemon in UNIX environments. It also has a TextPaneAppender, which will add the log messages to a JTextPane.

As you can see in Listing 1, each appender comes with a list of appender-specific settings (for instance, the RollingFileAppender needs to know the file path and name, the maximum size of file, number of backup files, etc., and the SocketAppender needs to know the remote host, port number, etc.).

Now that the problem of where to log is solved, the appender also needs to know the layout in which it should log. Hence all appenders have a layout setting. There are different types of layouts, such as SimpleLayout, TTCCLayout (deprecated), and PatternLayout (the most flexible one). PatternLayout can log time (in various formats), thread name, priority, category, time elapsed from 1970 to creation of logging event, and message. This appender can also log the class name, filename, location (method + class name + line number), line number, and method name, but these should be used only during development as they're very slow.

This completes the discussion about the log4j features. Let's talk a bit about performance. Look at this code:

```
if(cat.isDebugEnabled())
{
```

```
cat.debug("The account number is " + accountNumber);
}
```

isDebugEnabled will return true only if this category's debug messages will actually be logged. If not, it doesn't enter the if block, thus avoiding the cost of argument building. According to the documentation, on a 233 MHz ThinkPad running JDK 1.1.7B, it takes about 46 nanoseconds to determine if that statement should be logged or not. Actual logging is also fast, ranging from 79 microseconds using the SimpleLayout, to 164 ms using the TTCCLayout, and about a millisecond when printing exceptions. The performance of the PatternLayout is almost as good as the dedicated patterns, but with added flexibility.

How We Used It

We added a few of our own ideas to make it more effective. We derived our own exception class from RuntimeException to represent various exceptions in the system. The main feature of this class was that we put a log statement in the constructor to log the exception, the stack trace, and any nested exception. This logged every exception thrown in the system at any place, at any time with stack trace, and proved to be a very powerful tool.

Additionally, though PatternLayout provides facilities to log the class name and method name, it's supposed to be slow, so we mention these

Listing 1

```
log4j.rootCategory=WARN, ROOT_Appender
log4j.category.com.mycompany.employee=ERROR, EMP_Appender
log4j.category.com.mycompany.finance=INFO, FIN_Appender
log4j.additivity.com.mycompany.finance=false

# Set appender specific options.
log4j.appender.EMP_Appender=org.log4j.RollingFileAppender
log4j.appender.EMP_Appender.File=e:\\j dj\\employee.log
log4j.appender.EMP_Appender.Append=true
log4j.appender.EMP_Appender.MaxFileSize=10kb
log4j.appender.EMP_Appender.MaxBackupIndex=2
log4j.appender.EMP_Appender.layout=org.log4j.SimpleLayout

log4j.appender.FIN_Appender=org.log4j.RollingFileAppender
log4j.appender.FIN_Appender.File=e:\\j dj\\finance.log
log4j.appender.FIN_Appender.Append=true
log4j.appender.FIN_Appender.MaxFileSize=10kb
log4j.appender.FIN_Appender.MaxBackupIndex=9
log4j.appender.FIN_Appender.layout=org.log4j.PatternLayout
log4j.appender.FIN_Appender.layout.ConversionPattern=%d{dd MMM
yyyy HH:mm:ss,SSS} %-15.15t %-8.8p %-30.30c %x - %m\n

log4j.appender.ROOT_Appender=org.log4j.RollingFileAppender
log4j.appender.ROOT_Appender.File=e:\\j dj\\project.log
log4j.appender.ROOT_Appender.Append=true
log4j.appender.ROOT_Appender.MaxFileSize=10kb
log4j.appender.ROOT_Appender.MaxBackupIndex=9
log4j.appender.ROOT_Appender.layout=org.log4j.PatternLayout
log4j.appender.ROOT_Appender.layout.ConversionPattern=%d{dd
MMM yyyy HH:mm:ss,SSS} %-15.15t %-8.8p %-30.30c %x %C %F %l %L
%M - %m\n
```

Listing 2

```
import org.log4j.*;

public class LogDemo
{
    public static void main(String args[])
    {
        // Configure the logger using properties file.
        PropertyConfigurator.configure("e:\\j dj\\log.conf");

        // Get the logger for package "com.mycompany.hr"
        // as this is not exclusively configured, we will get
        // a new category inheriting the properties of the ROOT
        // category. The ROOT category is configured to
```

```
// log messages with priority "WARN" or higher. Hence
// "DEBUG" and "INFO" messages will not be logged.
Category logger = Category.getInstance("com.mycompany.hr");

if(logger.isDebugEnabled())
    logger.debug("A debug message");
if(logger.isInfoEnabled())
    logger.info("A info message");

logger.warn("A warning message");
logger.error("An error message");

// Get the category for package "com.mycompany.finance".
// This package is configured to log messages with priority
// "INFO" or higher, hence "DEBUG" messages will not be
// logged.
logger = Category.getInstance("com.mycompany.finance");
if(logger.isDebugEnabled())
    logger.debug("A debug message");
if(logger.isInfoEnabled())
    logger.info("A info message");

logger.warn("A warning message");
logger.error("An error message");

// Get the category for package "com.mycompany.employee"
// This package is configured to log messages with priority
// "ERROR" or higher, hence "DEBUG", "INFO" and "WARN"
// messages will not be logged. But also note that the
// additivity flag for this package is true (default setting).
// Hence the log requests will also go to the root package.
// So the "ERROR" message will also be logged in the "ROOT"
// category.
logger = Category.getInstance("com.mycompany.employee");
if(logger.isDebugEnabled())
    logger.debug("A debug message");
if(logger.isInfoEnabled())
    logger.info("A info message");

logger.warn("A warning message");
logger.error("An error message");
}
```



Batelle
N.W.
p/u

Cyscape
p/u

details in the message itself. For example:

```
Category cat = Category.getInstance(getClass().getName());  
cat.log("Account.getBalance: Total balance in account is " +  
accountBalance);
```

Here, when the message will be logged, we know that the class is `com.mycompany.finance.Account` and the method is `getBalance`.

Another helpful thing for database-specific applications is to create a wrapper around `java.sql.Statement` and log all the SQL statements fired. This is handy when the SQL statements are being dynamically generated.

We also had a unique requirement. In our system the same code was running for different companies. Think of it as if the accounting is being done for 10 companies. We wanted to be able to separate the log files when desired. But with the category named `com.mycompany.finance`, we couldn't do it. So we did a little modification and prepended the company name to the package name to get the category name. Our log statements looked like:

```
Category category = Category.getInstance(company + ".com.mycompany.finance");  
category.log("Account.getBalance: Total balance in account is " +  
accountBalance);
```

Now we not only know the company name for which this log statement is being executed, but we also have the ability to separate the log files by company name.

(A similar functionality can be achieved using Nested Diagnostic Contexts. The `log4j` provides a class called `NDC` for this purpose.)

One important and helpful thing we found was excellent support. Not only have bugs been fixed within hours of the first report, but suggestions have been received with an open mind. (By the time you read this article, however, the `log4j` package will be owned and maintained by Ceki Gülcü, the author of `log4j` package.)

Shortcomings?

There are a few shortcomings, but they're not limiting factors. One is that you can't remove a category once it's created. This was done to keep the API simple to use. Also, the appenders don't support JMS, XML, Database, or others. In the near future, more appenders like `JMSAppender`, `XMLAppender`, `IRCAppender`, `JDBCAppender`, and `JtableAppender` are planned.

Conclusion

Though other loggers are out there, we preferred `log4j` because it's powerful yet simple. However, this isn't a solution for everyone. You might want to check out other loggers such as Chris Barlock's `Jlog` (formerly called `JRAS`; it offers more in terms of filtering capability but at the cost of much higher complexity) and `GraceLog` by Chris Bladon (it enables you to inspect objects and log their contents).

Acknowledgments

I'd like to thank Ceki Gülcü, the author of `log4j`, for his invaluable input, and my friend Sumit Garg for helping me put this article together. I'd also like to thank Impetus Computing Systems and Info-One, whose resources I used to research and write this article. ☺

AUTHOR BIO

Vinay Aggarwal, an integration consultant at Trilogy in Austin, Texas, has extensive experience in Java and Web technologies. He received his degree in mechanical engineering from PEC in Chandigarh, India.

vinay_agg@yahoo.com

After the Connection

To understand JDBC, first get to know SQL

WRITTEN BY
ROBERT J. BRUNNER



In this series we've explored the process behind selecting a database and a JDBC driver as well as establishing a connection between your Java application and your database using JDBC. To actually do something useful, however, you need to be able to actively interact with a database using JDBC.

Early on in their history, relational database vendors agreed on a common interpretive language called SQL (Structured Query Language) that could interact with any database that supported this standard. This tactic helped build a large user base and a large number of third-party tools. Relational databases came to supplant the previously popular hierarchical databases as the dominant type of database system.

JDBC technology is built on this standard query language. As a result, to understand the fundamentals of JDBC, you must first understand the basics of SQL. Before delving into SQL, however, we need to cover some of the basic theory of relational database systems.

The two main concepts that underlie relational databases are the idea of a database transaction and the representation of data in a table format. Fundamentally, a transaction represents a logical unit of work with a database. In fact, the basic test for classifying something as a database, the famous ACID test that stands for Atomicity, Consistency, Isolation, and Durability, is defined in terms of transactions. If you wish to save your work, you need to commit your transaction; if you wish to undo your work, you need to roll back your transaction. Data locking, for both read and write operations, is used to prevent users from interfering with each other's transactions.

The data in a relational database is conceptualized to occupy a table (think of a spreadsheet) consisting of rows (records of data) of multiple columns (different types of data that make up the records). An entire theoretical framework has been developed, including such concepts as independent and dependent tables and different normal

forms, to prove theorems concerning relational database tables. For our purposes, however, we need to review only four concepts:

- **Primary keys:** Used to provide a distinct identity to each row in a table; essentially, they're why you always feel like a number when dealing with a particular bureaucracy (e.g., a call-number or an employee or Social Security number)
- **Joins:** Operations used to combine data from two distinct tables
- **Indices:** Used to facilitate rapid data retrieval; however, indices can occupy a significant amount of space and are therefore created only for a select number of columns in a table (generally including the primary key)
- **Views:** Results of given queries, often considered and treated as new tables

SQL Primer

At its most basic, SQL is used to interact with a relational database system in two different fashions, each with its own name: SQL data definition language (DDL) and SQL data manipulation language (DML). The former is used to create and drop tables from a particular *database*. The latter is used to select, insert, update, or delete data from a particular *table*. SQL is, by its very nature, an interpretive language that provides a great deal of flexibility but can also limit performance.

A few caveats are in order when writing or reading SQL. First, SQL commands are case insensitive and can be spread over multiple lines in an interactive SQL editor. An important footnote to this caveat, however, is that table and column names are generally case sensitive (as a point, I'll capitalize all SQL

commands and use mixed-case notation for all table and column names). This last issue is one of the biggest stumbling blocks for SQL novices as it's easy to forget. Second, SQL introduces standard data types that include representations for variable-length character arrays, integers, and floating-point numbers (see your database manual for more information). Finally, many interactive SQL tools use a special delimiter (such as the semicolon) to separate multiple SQL commands.

Before doing anything else, we need to create a table to hold our data. If we want to make a table to hold a simple list of contacts, we might use the following SQL statement:

```
CREATE TABLE Contacts (
  First VARCHAR(20) NOT NULL,
  Middle VARCHAR(20) NOT NULL,
  Last VARCHAR(20) NOT NULL,
  Phone INT,
  Age INT
);
```

This command creates a table named *Contacts* in the currently connected database. Its five columns are *First*, *Middle*, *Last*, *Phone*, and *Age*. The three name columns are variable-length character arrays with an initial size specification of 20 characters. These columns must contain actual data, indicated by the NOT NULL column qualifier. The last two columns are integer data types that are optional since they don't have the NOT NULL qualifier.

The complementary operation to remove a table from the database is considerably simpler:

```
DROP TABLE Contacts ;
```

After the necessary tables are created, the next step is to populate them with the relevant data. Many database systems have useful utilities that can automate the bulk inserts of large amounts of data; however, in SQL this is done using the INSERT command:

```
INSERT INTO Contacts
VALUES(
    'Robert',
    'J.',
    'Brunner',
    1234567890,
    21
);
```

Alternatively, we can explicitly specify the order of the columns that the value fields will occupy:

```
INSERT INTO Contacts (
    First,
    Middle,
    Last,
    Phone,
    Age
)
VALUES(
    'Robert',
    'J.',
    'Brunner',
    1234567890,
    21
);
```

Once we have a table filled with the data of interest, we can begin to flex the full power of relational databases. They can use the relations within the data to selectively query, update, and delete data according to specific constraints. Formally, we wish to perform a procedure (including SELECT, UPDATE, DELETE) on a table with a particular group of rows indicated by the appropriate relational operators (see Figure 1). For example, to display the full contents of our Contacts table (assuming we have inserted several entries), we can issue the following SQL command:

```
SELECT * FROM Contacts ;
```

While instructive, this example doesn't demonstrate the full power of the SELECT statement. Sometimes you may want to selectively extract certain columns for a subset of all the rows in a table (see Figure 2). For example, to pull out the First, Last, Age, and Phone columns for all rows that have an Age column value greater than 30, we'd use the following SQL command:

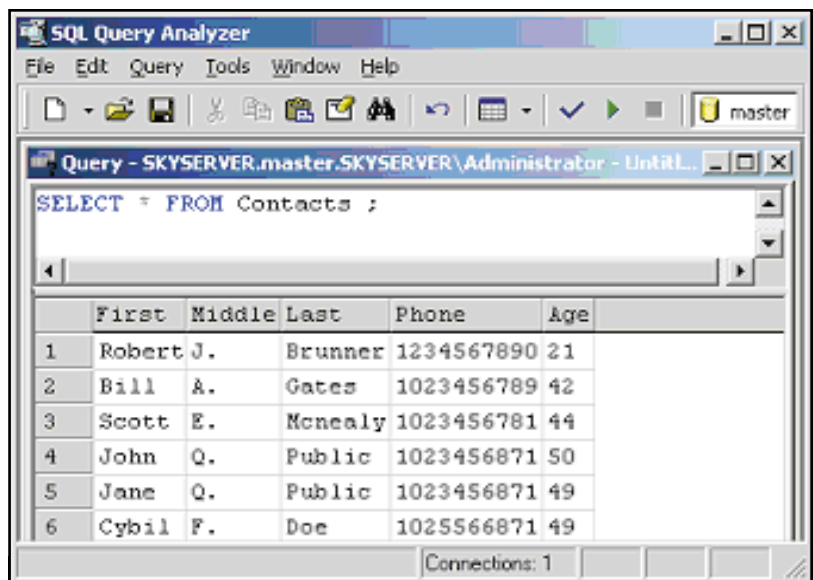


FIGURE 1 Result of the SELECT query using Microsoft SQL Server Query Analyzer Tool

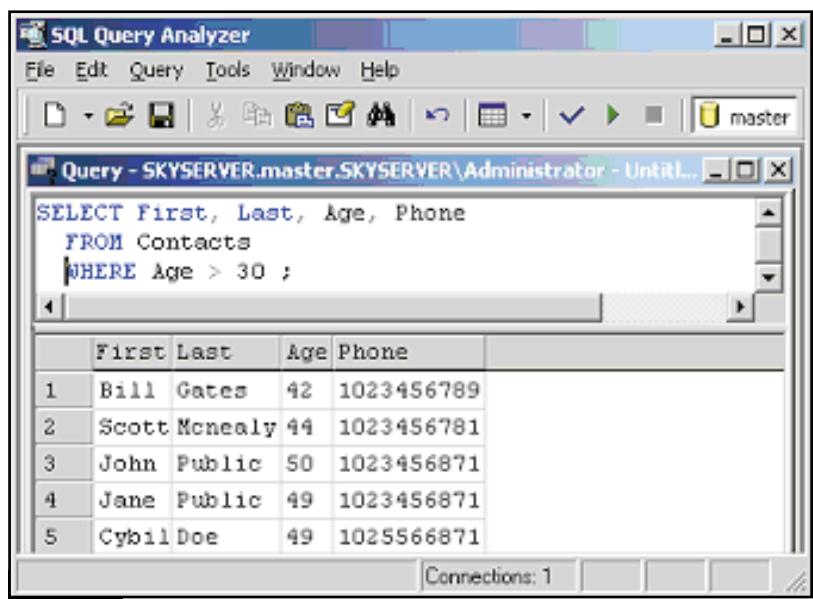


FIGURE 2 Result of the restricted SELECT query using Query Analyzer Microsoft SQL Server tool

```
SELECT First, Last, Age, Phone
FROM Contacts
WHERE Age > 30 ;
```

A similar syntax is used for the DELETE and UPDATE SQL commands, which either delete or update specific rows, respectively.

Morphing SQL into JDBC

Anyone serious about learning and using JDBC needs a good reference. I recommend the *JDBC API Tutorial and Reference, Second Edition: Universal Data Access for the Java 2 Platform* by Seth White et al. (Addison-Wesley), part of the Java Series. To understand why a good reference can be invaluable, recall that SQL has predefined data types, while the

Java language has its own predefined data types (one of Java's most important features). To pull data from a database into a Java application using SQL, you have to convert from SQL data types to Java data types and vice versa. The book referenced above devotes an entire chapter to this process, with many informative tables that demonstrate the allowed and recommended conversions.

Before delving into the more commonly used interfaces in the JDBC API, an introduction to the specific error-handling features is appropriate. Any time a database is involved, the whole concept of error handling can quickly become a quagmire. Fortunately, the JDBC API has provided an elegant solution – the SQLException object – which allows for chained exceptions, a novel concept in the Java

arena. As a result, the following convention is standard when using the JDBC API:

```
try {
    // JDBC Code
} catch (SQLException e) {
    while (e != null) {
        System.out.println("\nERROR:
\n");
        System.out.println("Message: "
+ e.getMessage ());
        System.out.println("SQLState: "
+ e.getSQLState ());
        System.out.println("ErrorCode: "
+ e.getErrorCode ());
        e = e.getNextException();
    }
}
```

Sometimes a database operation can produce a warning condition (encapsulated by the SQLWarning object), which is less severe than an exception but can also be chained. Since these conditions aren't exceptions, they're not handled in the typical try...catch block fashion. Instead, you need to check explicitly to see if a database operation generated a warning and act appropriately (see the previously mentioned JDBC book or the online Java tutorial, <http://java.sun.com/docs/books/tutorial>, for more information).

Now let's discuss using Java to interact with a database. Once a connection has been established with a database (see *JDJ*, Vol. 5, issue 10), the next step is to create a Statement object. This object encapsulates the process of passing SQL commands to the database and processing the results. As the code snippet below demonstrates, a Statement object is created from a Connection object, which in effect owns the newly created Statement object.

First we need the SQL command we want to send to the database. In this case it's our earlier table creation command, with the table renamed so we don't try to create a new table with the same name as an existing one (which would throw a SQLException).

```
String createString =
    "CREATE TABLE NewContacts " +
    "First VARCHAR(20) NOT NULL, " +
    "Middle VARCHAR(20) NOT NULL, " +
    "Last VARCHAR(20) NOT NULL, " +
    "Phone INT, " +
    "Age INT)";
```

Now we can create our SQL Statement object, which is obtained from an already established connection.

```
try{
    Statement stmt = con.createStatement();
```

Since this particular SQL string effectively updates the database and doesn't return any data, we call the executeUpdate method, which will execute the SQL command and return the number of rows affected (i.e., the update count).

```
int count = stmt.executeUpdate(createString);
```

```
// Process the Update Count
```

To properly release database resources (which are generally controlled by expensive license limitations), we need to close our objects properly:

```
stmt.close();
con.close();
```

```
} catch (SQLException ex) {
    // Handle Exception
}
```

This same syntax is used for SQL CREATE, DROP, INSERT, UPDATE, and DELETE operations.

On the other hand, SQL SELECT operations generate a new view of the data. To handle this data, the JDBC API uses the ResultSet object, which provides an interface to access the rows and columns of the resultant data. First we create the appropriate SQL string – in this case the previous SELECT statement – which returned certain columns for all rows with Age greater than 30:

```
String selectStatement =
    "SELECT First, Last, Age, Phone " +
    "FROM NewContacts " +
    "WHERE Age > 30";
```

The SQL Statement object is created as before, but now we'll call the executeQuery method, which will return the data resulting from our query. Notice that the ResultSet object is created from the Statement object; therefore, the Statement object effectively owns the ResultSet object. As a result, when the Statement object is closed or the object reference is reused, the ResultSet object is also closed.

```
try{
    Statement stmt =
        con.createStatement();

    ResultSet rs =
        stmt.executeQuery(selectStatement);
```

To access the resulting rows of our query, we use the built-in iterator of the ResultSet object to access all the rows pro-

duced by the query. This iterator is always initialized to point to the (fictitious) record that comes before the first record; thus, when it's first accessed (by the next method), it points to the first row:

```
while(rs.next()) {
```

To access the columns for each row in the ResultSet object, we need to convert them to their Java data types, which is done using the appropriate getXxx method, where we replace the XXX in the method declaration by the actual data type. For example, for the particular query we're using, we have two String objects followed by two integers. Assuming the appropriate variables have already been declared, we can easily extract the data:

```
firstName = rs.getString(1);
lastName = rs.getString(2);
age = rs.getInt(3);
phone = rs.getInt(4);
```

```
// Utilize the new data for some
processing
// For example, we could create
a new object,
// or print out all of the rows.
}
```

As before, you should close everything down properly so as to free up valuable database resources.

Of course, there's considerably more in the JDBC API than I've covered here. Some notable additions are the DatabaseMetadata and ResultSetMetadata interfaces, which can be used to obtain a great deal of information about the database or ResultSet, respectively, at runtime. I also didn't cover the PreparedStatement, which allows a database and JDBC driver to precompile particular SQL statements for improved performance or the CallableStatement interfaces, which allow a Java program to call a database stored procedure.

Conclusion

Hopefully this article has provided a gentle introduction to both SQL and the process of encapsulating SQL statements using JDBC. While I've intentionally avoided many of the details, you certainly have enough knowledge to get started, both with SQL and JDBC. With this basic understanding, you can begin to explore building components on top of the JDBC layer, as well as using applets, servlets, and JavaServer Pages to interact with a database. ☺

rjbrunner@pacbell.net

AUTHOR BIO

Robert Brunner is a member of the research staff at the California Institute of Technology, where he focuses on very large (multiterabyte) databases, particularly on KDD (Knowledge Discovery in Databases) and advanced indexing techniques. He has used Java and databases for more than three years, and has been the Java database instructor for the Java Programming Certificate at California State University Pomona for the past two years.

Eiffel-Like

SEPARATE CLASSES

Extending
Java's
concurrency
model with
asynchronous
objects

WRITTEN BY
FRANCISCO MORALES

To extend Java's concurrent behavior in a more natural way, in a more object-oriented point of view, we propose an extension to Java's concurrency model that will emulate Eiffel's separate statement. (Eiffel is an object-oriented language with a comprehensive approach to software construction.) The extension permits the attachment of nonphysical processors or threads to objects, thus allowing them to behave in an asynchronous and completely independent manner. This article briefly shows the concurrency tools of the Java programming language, points out their shortcomings, proposes solutions, and ends with the implementation of a solution.

It's useful to evaluate Java's concurrent programming model by reviewing how this language implements the concepts explained by Bertrand Meyer and referred to as "The three forces of computation," which represent the statement: "To perform a computation is to use certain processors to apply certain actions to certain objects."

Java classes and objects play the same role in Eiffel as they do in Java. In Java the notion of a nonphysical processor fits in with the concept of thread as represented by the Thread class. A processor would then represent an autonomous thread of control that's capable of supporting the sequential execution of instructions on one or more objects.

Differences arise when we look at the possible assignment of processors to actions. In Java we have only the run() method, which belongs to the Runnable interface, for concurrent execution. This interface must be implemented by a class attached to a Thread instance in construction time and then, and only then, can we explicitly start the thread's execution, employing the technique known as delegation.

The problem originates from the fact that threads and objects represent different entities – methods run on threads, objects do not – so the only way to get a method to run concurrently (in an independent thread) is to call it from inside that thread's run() method. This is a disadvantage of this model since the object technology's basic model for computation would somehow be broken, as we can easily conclude from the following statement:

```
objRef.operID(args);
```

Here the execution of method operID(), called as part of an action performed on a client object, won't occur concurrently as long as it's not placed inside a thread's run() method. The model is said to be broken since there's no natural, object-oriented way in Java to provoke the attachment of different streams of execution to both the client object and objRef. The entire class design should be oriented to the implementation of the Runnable interface and the creation of Thread class instances.

Figure 1 shows, in UML notation, a possible general pattern for this problem.

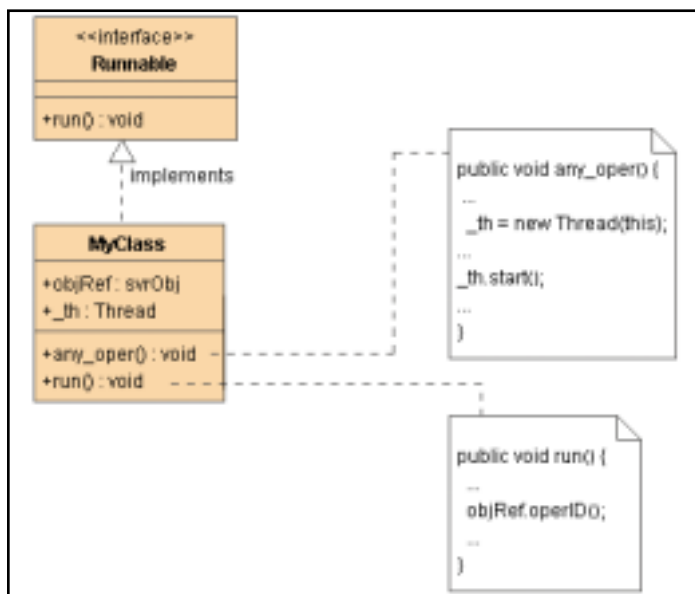


FIGURE 1 A general pattern for threads creation in Java

The benefits of this mechanism are based on the fact that Java clearly separates the notion of data abstraction, implemented by classes, from the notion of control, represented by the framework that's composed of the Thread and ThreadGroup classes and the Runnable interface. Table 1 shows the main components of this framework.

CLASS/INTERFACE	ROLE IN CONCURRENT JAVA PROGRAMMING
Class Thread	Abstraction of an independent flow of control
Class ThreadGroup	Hierarchical grouping of threads for security and easy handling
Interface Runnable	A way of defining chunks of code to be independently executed. The core Java concurrent execution mechanism relies on associating a Thread instance with an instance of a class that implements this interface.

TABLE 1 Java's concurrency tools

Once this model has been analyzed, let's proceed with Eiffel's concurrency model.

A Closer Look at Eiffel's Model

We'll now analyze Eiffel by looking for features that are capable of improving the former concurrency model. According to Meyer: "... any object O₂ is handled by a certain processor, its handler; the handler is responsible for executing all calls on O₂ (all calls of the form x.f (a) where x is attached to O₂)."

From this statement we can conclude that the proposed model has two different call semantics:

- **Synchronous:** The client object will be forced to wait for the server to complete its operation, as requested in the code.
- **Asynchronous:** An object doesn't need to wait for the others to proceed with its execution, since it occurs in different processors.

In Java the default semantic is obviously synchronous. It doesn't provide any syntactic construction in its classes that will specify a different processor for its methods. So we need to define asynchronous execution through the use of Thread instances and the method run. This is the only way to define an asynchronous split of control.

Defining Separate Entities in Java: A Straightforward Implementation

The essential difference between sequential and concurrent execution of actions is that the handler of the call's target is different from the one that originated the call. Doug Lea's interactive diagrams show this difference with solid and dashed lines that represent synchronous and asynchronous method calls. Figure 2 depicts such an interactive diagram for the pattern in Figure 1.

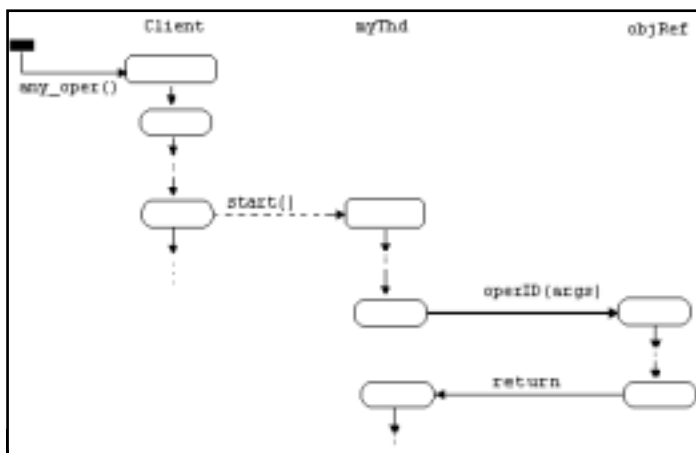


FIGURE 2 An interactive diagram

To implement this difference in Java, we'll follow Eiffel's approach, which allows us to define separate entities in two ways. The first one permits the creation of an object's instance and attaches it as an independent processor.

```
x: separate A
```

Here we're declaring an object x, instance of class A, that you'd execute on a different processor.

The second approach takes a static form and is applicable when all the instances of a class are intended to be separate entities:

```
separate class A ...
```

We cover both forms in this article. Our semantics require a new processor to handle all the messages for each of the instances of our separate classes or objects.

Obviously, we need to modify the Java syntax, adding a new keyword as an optional modifier in class and field definitions. To accomplish this, a preprocessing approach is proposed. This means that a parser program should be developed for parsing the Java source file and eliminate, if included, the separate modifier and replace it with some standard code.

The following is our first approach to the problem, but not the best one. We'll add some code before any invocation to a public method, and in separate-declared classes replace every method's body with code that's responsible for the creation of a new thread and the execution of its run() method. Inside this method we'll select the original code and invoke it. That's not the exact semantic of *separate* in Eiffel, but it provides us with a good place to start our evaluation. Since we need to intercept method invocation, we use the Java Reflection API.

Java Tools for the Solution

The Reflection API is composed of a set of classes that enable us to discover information about Java classes at runtime. This new feature of Java, developed with version 1.1 of the JDK, has also been called the *introspection API* because it gives objects the ability to look inside themselves or other classes during runtime.

The API defines the following elements:

- At the core of the API is an object called *Class* for reflecting classes and interfaces in a running Java application.
- Every object has a constructor, can perform a set of actions (also called methods), and is characterized through a number of attributes or variables. All these components are reflected through the Constructor, Method, and Field objects in the API that permit us to obtain information about each object's elements according to the security police used.
- The Member interface, implemented by the three former objects, contains the prototypes of methods that allow you to query the object's members.
- To represent primitive Java data types, the API contains nine Class objects defined as constants such as java.lang.Boolean.TYPE and java.lang.Character.TYPE.
- There are two utility classes: Array and Modifier. The former allows us to access and construct Java arrays dynamically; the latter helps to decode language modifiers on classes and members.

Let's now proceed with our desired output definition on the basis of these elements.

Implementing the Solution

Separate as a Class Modifier

Let's now define how our solution should work, what we need as input data, and what we should produce. First we have a class like the one in Listing 1. After preprocessing, we obtain the code in Listing 2.

Two things must be taken into account:

1. The security police used
2. The methods' formal parameters

The security police that are used could affect the way we obtain the original method references, the one we invoke later in method run(). In Listing 1 we defined the method m₂() as public, since the getMethod() method just returns Method objects that reflect the specified public member of the class. To obtain a reference to members affected by private-, protected-, or default-access modifiers, like m₁(), for example, we can take one of the two approaches listed below:

1. We can use the getDeclaredMethod(...) method instead of getMethod(), which can throw a security exception when a security manager is installed.
2. As in Listing 2, the original modifiers on the methods generated by the preprocessing parser, m₁() and m₂(), remain untouched, and the original programmer's methods in the generated code, orig_m₁() and orig_m₂(), get declared public. This approach shouldn't affect the security of the application, since orig_m₁() and orig_m₂() are completely new for the program and invocations on them don't exist.

The second item to take into account is the methods' formal parameter treatment that must be done. The reflection API allows us to refer them at runtime through an array of Object instances, the __params variable used in Listing 2. Because of this, you shouldn't declare any parameter belonging to some of Java's primitive types. Instead we propose the use of the corresponding wrapper classes found in the java.lang package such as Integer, Float, and Double.

Separate as a Modifier for Class Fields

This solution is a little bit trickier since we'll find situations where the object's instances should run in their own thread and others where this behavior won't be needed.

To accomplish this, we've implemented the desired output in an inner class whose role is to permit the execution of each method's invocation in an object's thread. We use the starting code in Listing 3, then after preprocessing we should obtain the code in Listing 4.

As you can see, the changes affect classes A and B. The _Aux class in A encapsulates the behavior to execute in the separate case. Each invocation of m₁() in B should be replaced with a separate_m₁()

Another detail to take into account is the use of the getDeclaredMethod() method, instead of the getMethod() mentioned earlier, for taking a reference to the method that will be invoked. The main reason behind this decision is that we're invoking methods of A from class B, so we can't make them all of public type.

One more thing: the solution of encapsulating the separate behavior in an inner class is perfectly portable to the first analyzed case. The difference is just in the semantics of the problem.

Implementing the Preprocessing Parser

Traditionally, as everyone knows, a parser is a tool capable of performing three basic tasks:

1. It receives some text from an input source, a Java program, for example.
2. The text has to be organized according to some predefined criteria.
3. Some actions have to be performed on this input based upon the above-generated organization.

The first two tasks are frequently encompassed and performed by an auxiliary application named *scanner*. Its main objective consists of building high-level language units called *tokens*, while discarding constructs that don't represent any useful information, such as white spaces and comments. Figure 3 depicts this functionality.

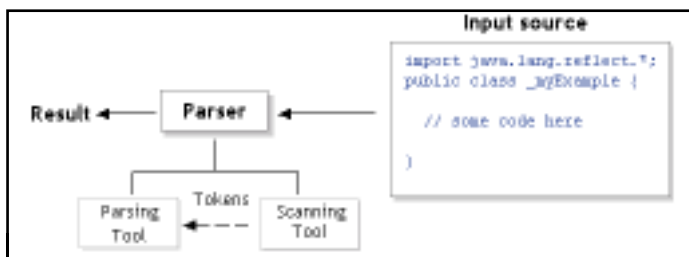


FIGURE 3 Parser's functionality

The actions mentioned in Step 3 try to match the tokens generated by the scanner against a language specification. If this match becomes successful, the parser then performs some action upon the construct. In our case it could generate the code that implements the separate behavior.

Building a parser from the ground up is a highly complex and time-consuming task. We used a parser generator, namely Sun's Java Compiler Compiler version 1.0, or JavaCC, which can be freely downloaded from Sun's Web site or from Metamata.com. The last one offers the parser generator and a set of example grammars.

A parser generator is usually a tool that's capable of generating parser and scanner programs starting from a grammar specification of the language to one we want to parse. The Java grammar we used was one of the samples that ships with JavaCC from Metamata.com, which covers Java's 1.1 language extensions.

JavaCC is based on ANTLR technology; also, it is an LL(k) predicate-based parser generator. It is beyond the intention of this article to describe in depth the details of the parser generator's algorithms. The interested reader should note the references at the end of this article.

Changes to the Java Input Grammar

To correctly parse a Java program whose objects can be affected by the separate modifier, first modify the Java grammar that's used as input for the generation of the parser and scanner tools. For this purpose there are only a few elements to modify from the original grammar construct:

- The set of reserved words and literals that have to include the token separately
- The set of class and class field modifiers that allow *separate* to be a valid modifier for both elements

These changes are very simple since the specification language to build the input description is Java-based with a few extensions to specify grammars (see Listings 5 and 6).

Note: There's a special method for analyzing a class field, instead of a general method for fields belonging to classes or interfaces. That's because the interface's fields may not be declared separate.

Output Generation

The generation of the output code, in case of separate behavior, is straightforward. In the `JavaParser` class generated by JavaCC, when the separate modifier is found, we start storing the class elements defined by the programmer since the parser just analyzes language units, eliminating the separate token. At the end of this analysis the stored code is added to the one needed for separate behavior, which is generated by the class `JavaParserNewCode`.

To accomplish this generation, the only things to take into account are the method definitions, since separate behavior consists only of executing every method invocation in its own processor.

JVM Scheduling vs Eiffel's Concurrency Control Files

We've defined processors as abstract concepts, independent of the underlying hardware and operating system architectures. In doing so, we've achieved a desirable independence for physical details. Our object system can run on a single processor workstation with a time-sharing schedule or in an SMP server.

Eventually we need to assign our physical resources to the processors of our program. In Eiffel we have the Concurrency Control File as a mechanism for defining this mapping.

In Java it's impossible to define such things programmatically. Threads are created dynamically in runtime as any other object, and the scheduling is left to the Java Virtual Machine (JVM). We can use groups of threads and priorities to obtain some level of control over the execution of threads, but the core algorithm is embedded in the JVM and, in current versions of the JDK, it's not possible to customize or substitute. In fact, it's somewhat implementation-defined (as noted in earlier versions of JDK, Solaris and Windows implementations differ in that the first is not preemptive).

In Java, distributed programming uses an additional API, called *RMI*, for communicating remote JVMs via remote interfaces. Alternatively, we can use IDL and CORBA.

The JVM is an operating system process in current commercial OSs, so they're under control of the OS process control. We haven't heard about parallel implementations of the JVM, exploiting the runtime information about threads, but it's clear that it would be advantageous to obtain better performance in architectures that include multiprocessing.

Conclusion

We've described our experiences and related studies regarding the introduction of Eiffel's concurrency model into Java. Our goal has been

to extend Java's concurrent programming capabilities with the creation of completely independent objects through the use of a separate modifier.

Our approach helped us to unify, in some way, the notion of objects and threads, giving a more consistent object-oriented view to the design and implementation of concurrent applications in Java. The newly introduced model keeps the conceptual architecture strictly separate from the physical one; the former assumes that there'll be as many resources as the application needs; the latter is responsible for the creation of threads and their assignment to separate objects.

Another benefit to this approach is applicable to the academic field; we've taught Concurrent Java Design for three years and lacked a uniform object-oriented-based vision of the problem. With our extension, the design of concurrent Java applications is divided into two stages. The first is based on objects that are affected by the separate clause in which we apply the usual object-oriented execution mechanism of `object.operation(args)` and introduce the concurrency concepts in a more natural way. In the second stage we present Java's concurrency tools in more detail, going deeper into its mechanism and behavior.

Through this division, new concepts become simpler and more natural for the student since the move from sequential to concurrent programming just represents, initially, a little change in processor assignment.

One problem is the management of distributed resources. In Eiffel resources are mapped to processors through the Concurrency Control File, which enables you to manage the allocation of local and remote resources to processes. This unified view of mapping creates a better plan for running threads, allowing applications to control the physical allocation. The only ways for thread control in Java are, as mentioned previously, the `Thread` and `ThreadGroup` classes and the methods inherited from the `Object` class. These elements have to be combined with the functionality contained in the *RMI* package when we're developing distributed multithreaded applications. This makes this work error-prone and creates hard-to-predict client machine dependencies. Hence, we're preparing an extension of Java's thread control capabilities with support for distributed resources management. ☺

References

1. Lea, D. (1997). *Concurrent Programming in Java. Design Principles and Patterns*. Addison-Wesley.
2. Meyer, B. (1997). *Object-Oriented Software Construction*. Prentice Hall.
3. "ANTLR: A Predicated-LL (k) Parser Generator." *Software – Practice and Experience*, Vol. 25(7), 789–810. July 1995.
4. Voss, G., and Parr, T. "Parsers, Part I." (<http://developer.java.sun.com/developer/technicalArticles/Parser/SeriesPt1/index.html>). January 1997.
5. Voss, G., and Parr, T. "Parsers, Part II. Building a Java Class Browser." (<http://developer.java.sun.com/developer/technicalArticles/Parser/SeriesPt2/index.html>). January 1997.
6. Parr, T., and Coker, J. "Parsers, Part III. A Parser for the Java Language." (<http://developer.java.sun.com/developer/technicalArticles/Parser/SeriesPt3/index.html>). March 1997.
7. Stanchfield, S., and Parr, T. "Parsers, Part IV. A Java Language Cross-Reference Tool." (<http://developer.java.sun.com/developer/technicalArticles/Parser/SeriesPt4/index.html>). December 1997.
8. Joy, B., Steele, G., Gosling, J., and Bracha, G. *The Java Language Specification*. Addison-Wesley.

AUTHOR BIO

Francisco Morales, an assistant professor at the Pontifical University of Salamanca in Madrid, is also an independent software consultant. He holds a B.Sc. and M.Sc. in mathematics from Dresden's Technological University in Germany.

fmorales@wanadoo.es

Listing 1: Sample class code for later processing

```

separate public class A {
void m1(){
    //... m1's code
}

public void m2(Integer x) {
    // ... m2's code
}

    // ... Other class' code
}

```

Listing 2: A straightforward implementation of separate behavior attached statically to a class

```

import java.lang.reflect.*;

public class A implements Runnable {
    // Data needed for each method invocation
    private Method _method; // Method to invoke
    private Object[] _params; // Method's parameters
    private Object _ret; // Return value

    public void run(){
        try {
            _ret = _method.invoke(this,_params);
        } catch(Exception e){
            System.out.println("Error: " + e.getMessage());
        };
    }

    // Methods to execute the separate way
    synchronized public void m1(){
        Class [] FormalParams = new Class[0];
        _params = new Object[0];
        try {
            _method = this.getClass().getMethod("orig_m1",
                FormalParams);

        } catch(NoSuchMethodException e){
            System.out.println("Error: " + e.getMessage());
        };
        new Thread(this).start();
        return;
    }

    synchronized public void m2(Integer x) {
        Class [] FormalParams = { x.getClass() };
        _params = new Object[1];
        _params[0] = x;
        try {
            _method = this.getClass().getMethod("orig_m2",
                FormalParams);

        } catch(NoSuchMethodException e){
            System.out.println("Error: " + e.getMessage());
        };
        new Thread(this).start();
        return;
    }

    // The original programmer's methods
    public void orig_m1(){
        // ... m1's code
    }

    public void orig_m2(Integer x){
        // ... m2's code
    }

    // ... Other class' code
}

```

Listing 3: A Java class applying separate to a class's field

```

class A {
    public void m1(){
        //... m1's code
    }
}

class B {
    public static void main(String[] args) {
        separate A _a = new A();
    }
}

```

Listing 4: Separate behavior implementation for a class's field

```

import java.lang.reflect.*;

```

```

class A {
    // Elements for SEPARATE behavior attached to
    // fields.
    synchronized public void separate_m1() {
        try {
            new _Aux.callMethod(this.getClass().getDeclaredMethod("m1",
                new Class[0]),
                new Object[0]) ; ???FMA - This
        } catch(Exception e) {
            System.out.println("Error: " + e.getMessage());
        }
    }

    public class _Aux implements Runnable {
        private Method _method;
        private Object[] _params;
        private Object _ret;

        public void callMethod(Method _method,
            Object[] _params) {
            this._method = _method;
            this._params = _params;
            new Thread(this).start();
        }

        public void run() {
            try {
                _ret = _method.invoke(A.this,_params);
            } catch(Exception e){}
        }
    } // End of the inner class

    // Original class code
    public void m1() {
        // ... m1's code
    }
}

```

```

class B {
    public static void main(String[] a) {
        A _a = new A();
        _a.separate_m1();
    }
}

```

Listing 5: Separate added to the set of the language's reserved words and literals

```

TOKEN :
{
    ... // All Java tokens plus ...
    < SEPARATE: "separate" >
}

```

Listing 6: Changes made to the Java language grammar

```

...
void TypeDeclaration() :
{
    LOOKAHEAD( ( "abstract" | "final" | "public" |
        "separate" ) * "class" )
    ClassDeclaration()
    |
    InterfaceDeclaration()
    ";"
}

void ClassDeclaration() :
{
    ( "abstract" | "final" | "public" | "separate" ) *
    "class" <IDENTIFIER> [ "extends" Name() ]
    [ "implements" NameList() ]
    "{" ( ClassBodyDeclaration() ) * "}"
}

void ClassFieldDeclaration() :
{
    ( "public" | "protected" | "private" | "static" |
        "final" | "transient" | "volatile" | "separate" ) *
    Type() VariableDeclarator() ( ","
        VariableDeclarator() ) * ";"
}

```



A Case Study: Extreme Programming with EJB

Under the right conditions, EJB and XP are a powerful combination

WRITTEN BY
JASON WESTRA



This month's EJB Home was originally a presentation at JC2 in Santa Clara, California, in September. For those of you who couldn't make the session, I thought it would be beneficial to transcribe it here and relay an experience in the successful implementation of an EJB application using XP.

EJB/XP Overview

I attended a Boulder Java Users Group in August. The topic was XP. The speaker asked, "How many of you know what XP is?" and half the attendees raised their hands. Next, he asked, "How many of you have practiced XP on a project?" and half of the hands dropped. The next question he should have asked was, "From those of you remaining, how many have practiced XP on an EJB development effort?" Quite frankly, I don't think many hands would've remained!

XP is a methodology for software development that turns previous methodologies on their heads. It takes commonsense software development ideas, such as code reviews, unit testing, and integration testing, to the extreme while maintaining some general goals. These goals are risk management, producing high-quality software, building real applications (not massive frameworks), and, last but not least, having fun!

To accomplish these goals, XP advocates a number of "enablers," including simple designs, an iterative

development process with pairs of programmers working side by side, and a heavy emphasis on unit testing (see Figure 1).

Enterprise JavaBeans is an integral API in the J2EE platform and the foundation of Sun's approach to server-side component models. EJB has been taking the software development industry by storm, especially in the e-commerce space, where *n*-tiered transaction-processing systems reign. EJB offers many features that enable your team to accomplish the goals of XP.

Both EJB and XP are relatively new, and development organizations are often reluctant to take on too many paradigm shifts in their development process at one time. In general, this caution is warranted; however, I think that EJB/XP play together well. Let me describe why I think they're a good combination.

EJB/XP: "Hand in Hand"

Simple Designs

XP advocates keeping designs as simple as possible. This prevents developing code that's feature rich, yet useless for solving your present business problem. In other words, don't over-architect your system! The J2EE platform is an excellent way to keep your designs simple. With J2EE the architecture of your system is already defined, eliminating the need to create blueprints of how each tier will interact.

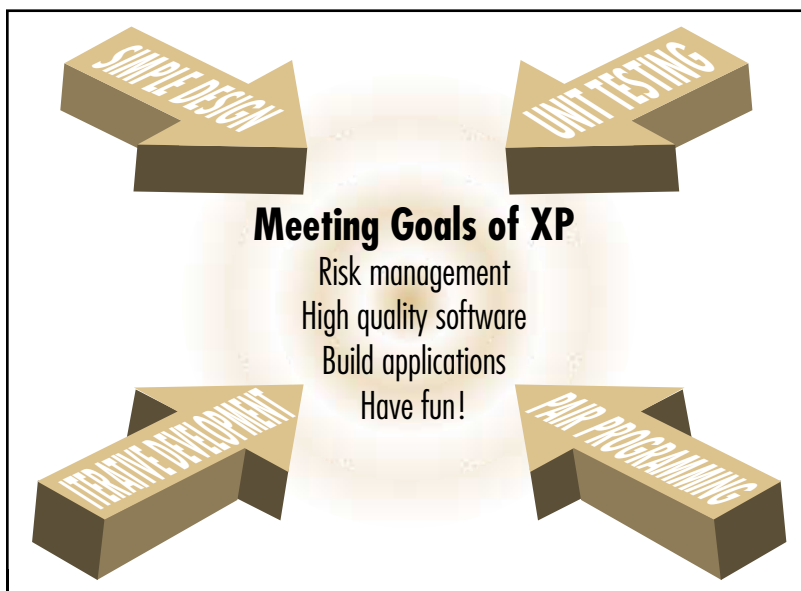


FIGURE 1 Accomplishing XP goals

While EJB development can be complex, many standard EJB design patterns exist to foster quick development and eliminate the need to reinvent these designs. EJB design patterns have been published on many sites; one that includes an expansive list is www.the-serverside.com, an offering monitored by Ed Roman, author of *Mastering Enterprise JavaBeans and the J2EE Platform*.

Last, EJB provides several features that ease the pain of designing distributed transaction semantics, persistence, and security in your applications. In particular, EJB offers CMT (container-managed transactions), which eliminates the need to design and code transaction demarcations in your application. Similarly, CMP (Container-Managed Persistence) allows you to develop database-independent entity beans in your persistence layer without worrying about designing to support multiple DB vendors. Also, EJB supports declarative security, which eases the burden of designing role-based security into your application. In general, the declarative nature of the EJB component model fosters simple designs, an added bonus on an XP project.

Iterative Development

XP advocates developing software with an iterative approach. Iterative development is nothing new; taken to the extreme, it means releasing solid code every one to four weeks during a project. EJB has a few tricks up its sleeves to foster iterative development in an EJB/XP environment.

1. EJB's component model allows for declarative transactions, persistence, and security. Declarative properties ease changes to your application by eliminating the need to update code to modify application logic.
2. Once an EJB is developed, it's packaged into an `ejb-jar`, which is a single, deployable unit. This approach allows you to iteratively release changes to your EJBs in a controlled manner. `Ejb-jars` can be versioned to ensure current releases are correct.

Unit Testing

XP emphasizes unit testing to the extent that you should write your tests even before you code. This practice is often taken with a grain of salt, yet unit testing your EJBs with a strict methodology is critical. There are numerous areas where XP's emphasis on unit testing enriches your EJB experience.

Establishing a unit-testing methodology allows you to quickly test upgrades to your code and J2EE server

when they support new versions of the EJB specification. Also, unit testing across tiers in an EJB application enables you to track errors more quickly.

Testing responsibilities can be mapped quite well to EJB roles defined in the EJB specification. In particular, it is the responsibility of the bean provider to test any components that he or she has developed. System administrators of an EJB/XP project should manage versions of EJB deployment units and run daily builds on source code to ensure that it compiles and builds into compliant `ejb-jars`. An EJB/XP project's system administrator should run full test suites against code daily (at least) to make sure broken EJBs haven't been checked into source code control.

Pair Programming

One of the most controversial pieces of the XP methodology of software programming, pair programming is the technique whereby two developers share a single development environment, including a single monitor, keyboard, and mouse. While one developer is coding, the other is actively thinking about test cases and watching to catch errors sooner rather than later in the development cycle.

EJB development is complex, and the tasks involved to develop an EJB are numerous and often forgotten. For instance, the numbers of windows open while developing and deploying EJBs can be daunting. At any time on your desktop you may have the following open: your EJB server console window, your IDE (integrated development environment), a console for building your EJBs, a database management window, and a third-party debugger window! An extra pair of eyes speeds the development and code quality of your EJBs as you sift through the layers of windows.

I've introduced EJB and XP as a powerful combination to develop distributed applications. Let's see if they're really a viable mix by investigating how CQG, Inc., adopted the two ingredients into a successful recipe.

Case Study: CQG, Inc., and EJB/XP

The following section describes how CQG, Inc., a real-time, graphic-enhanced quote vendor, utilized EJB/XP successfully to complete an application in less than five weeks.

Company Background

CQG receives data from futures and stock exchanges around the world and distributes it to traders in more than 50

countries. CQG has a reputation for producing the cleanest and most reliable data in the industry, and it must be available 24x7 with zero downtime. The company is headquartered in Denver, with branches in Chicago, New York, London, Paris, Frankfurt, Moscow, Milan, Zurich, and Tokyo. CQG's development centers are based in Denver, Boulder, Moscow, Dallas, and London.

CQG's legacy systems had become monolithic and unmanageable. Refactoring wasn't easy since the person who had developed the original source code was generally unavailable when enhancements were needed. CQG wanted to rewrite and enhance legacy systems with new, component-based technology for future maintainability, performance, expandability, and flexibility to change.

In early 2000 Ken Goodhew, a project development manager for CQG, and his team, working out of the Denver and Boulder locations, were tasked with the redevelopment effort. Under Ken's guidance, CQG followed XP-like practices for years, including small releases, simple designs, and small teams of developers. CQG also unit-tested code initially, but seldom maintained the tests as code evolved. Ken wanted to apply Java and XP to the effort in as many ways as possible; however, his team was relatively new to Java. Most of them were familiar with C, C++, and Perl on a UNIX platform, and none had applied XP in a strict fashion before.

After identifying seven stories that it wanted to convert (a *story* is an XP concept representing a piece of functionality the customer wants in a system), CQG decided the J2EE platform would be the foundation on which to build the new applications. Under Ken's guidance CQG also decided that it would pursue the XP paradigm in its new endeavors. The missing piece of the equation was a solid mentor who possessed experience in both EJB and XP to ensure the success of CQG's first implementation. CQG enlisted Verge Technologies Group, Inc., based in Boulder, to help develop a prototype application using EJB/XP concepts and perform knowledge transfer in the J2EE platform, BEA's WebLogic Server, and XP.

Enabling XP for EJBs: The EJB/XP Environment at CQG

Based on Verge's experience with EJB/XP, the development environment is the heart of a successful EJB/XP. In particular, the IDE should be geared toward server-side development and a

solid unit-testing framework. Methodology must be established, and the environment must support quick deployments of EJBs to unit-test them painlessly after minor changes have been made. Figure 2 reflects the tools selected for CQG's EJB/XP environment.

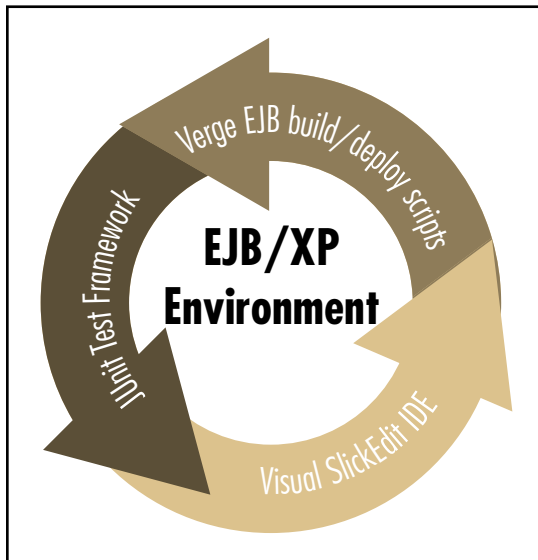


FIGURE 2 CQG's EJB/XP environment

The IDE chosen for the EJB/XP environment at CQG was MicroEdge's Visual SlickEdit (www.slickedit.com). SlickEdit offers several features that are conducive to EJB/XP development, including tagging, which allows you to set up the IDE to auto-complete your code. Auto-complete helps developers come up to speed with new J2EE APIs by presenting them with available choices for packages, classes, methods, parameters, and attributes while inside its editor. This is helpful for rapid development because you don't have to consult the J2EE API Javadocs for the information. SlickEdit's tagging feature also lets you tag only the packages you're interested in; thus you can leave out all AWT and Swing packages that aren't necessary for server-side development.

Other convenient features for EJB/XP that SlickEdit offers are multiformat support and multiplatform support. The former allows you to develop your EJB source code (.java) and then edit the deployment descriptors (.xml) from the same environment. You can edit shell scripts, DDL, DOS scripts, and Perl scripts from SlickEdit as well. It supports both Windows and Linux, so development is possible on either of these platforms, a bonus for Java development in general!

JUnit (www.junit.org), an open source testing framework for Java, was chosen for the unit-testing framework at CQG. JUnit provides a pattern-based

approach to developing individual unit tests and including them in larger test suites.

Verge introduced CQG to its methodology of unit-testing EJBs. In this methodology a JUnit test was developed to test individual entity beans first. This ensured that data layer access was functioning correctly. Once this tier was functional, the business logic tier of session beans was tested. Session beans that supported critical business functions were tested to ensure that they were supported accurately. Errors found at this level were usually attributed to business logic errors since the entity layer was determined to be solid from previous unit tests.

In this methodology, once a number of individual unit tests are created, a "test suite" is formed that can be run from start to finish against the system. Suites at CQG accessed a test bed of data or cleaned up after themselves to ensure data generated by the tests wasn't left over in the database.

Finally, to automate builds and deployments of EJBs, Verge introduced its suite of Perl scripts to CQG's EJB/XP environment. The use of GUI tools to deploy EJBs isn't conducive to EJB/XP programming. Such tools are memory hogs and take longer than single command-line script to build and deploy an EJB. Also, a system administrator can execute a process to build all EJBs and run test suites against them without having to build each EJB through a GUI. Once an application consisting of 10 or more EJBs is in production, a GUI isn't a viable option, and I recommend using scripts for any environment whether or not it's an EJB/XP project.

There are numerous options for creating scripts for building and deploying EJBs. A DOS script can work fine for NT development and production environments; however, it won't be portable when your environment needs include UNIX. The Verge scripts that constituted CQG's EJB/XP environment are written in Perl for cross-platform portability. This ensures that an EJB/XP environment remains consistent (e.g., using same scripts, classpath settings, server settings, etc.) across UNIX and NT, and it prevents the need to update different scripts for each platform.

What successes came from the combination of EJB and XP at CQG? The following section describes some of the pieces that CQG found most valuable in its initial trek into EJB/XP.

Success Stories of EJB/XP at CQG

CQG's first attempt at EJB/XP programming was a success. A proof-of-

concept prototype with both a Swing and JSP front-end accessing EJBs was developed in five weeks using the EJB/XP paradigm. Ken Goodhew attributed this success to two critical factors: relentless unit testing fostered by the EJB/XP environment and pair programming.

The JUnit tests developed by the team gave CQG the confidence to refactor the existing code base, adding enhancements as needed. The test suites also proved valuable when CQG attempted to upgrade to a new release of their chosen application server. The tests failed on several accounts because of a bug in the server's new version, leading CQG to wait for a service pack to provide the fixes before migrating.

Pair programming enabled CQG to develop EJBs quickly and to learn from the effort as well. Verge passed on its EJB experiences to the CQG team by pairing a CQG developer with a Verge consultant at all phases of the project from EJB/XP environment setup to JSP and EJB development. In typical pair programming fashion, they shared a keyboard, mouse, and monitor and sat shoulder to shoulder at tables in a common development room. At the end of five weeks, ownership of the application was not held by a single CQG team member, but was collectively owned by the group.

Combining EJB's and XP's strong points, such as a component-based model, and XP's emphasis on unit testing and collective ownership through pair programming, allowed CQG to be confident in its ability to maintain and enhance its new applications.

Conclusion

EJB and XP are a powerful combination under the right conditions. Your development environment must enable rather than restrict good XP practices, such as many small releases and iterative development, and unit testing. You should emphasize pair programming to foster knowledge transfer and code quality of your EJBs. Take advantage of EJB features, such as declarative security, transactions, and container-managed persistence where applicable in your system, to ease refactoring your logic. In addition, staff your team with EJB/XP experts to avoid attempts at solving problems that have already been solved and to transfer knowledge through pair programming. With the right recipe your first EJB/XP experience will be as enlightening as CQGs. ☘

westra@sys-con.com

AUTHOR BIO

Jason Westra is CTO at Verge Technologies Group Inc., a Java consulting firm specializing in e-business solutions with Enterprise JavaBeans.

WRITTEN BY
SHERRY SHAVOR, PETER HAGGAR, AND GREG BOLLELLA

Given the popularity of the Java software application development platform and the market potential for embedded devices, there's a need to understand how programs can take advantage of the use of Java for embedded application development. This article investigates three Java tools that are available for the embedded engineer and analyzes how well they map to the needs of the embedded community.

This article draws on the results from a study of the emerging embedded Java marketplace. The material was gathered by a team whose skills included a solid knowledge of object-oriented programming, the Java programming language, and embedded and real-time programming. It focuses on the following three products:

- IBM VisualAge MicroEdition
- Microware PersonalJava for OS-9
- Sun KVM

The functions and features of these products as well as the overall lessons learned about the embedded Java environment are also described.

A variety of
solutions are
available to
provide Java
support in the
embedded space

A LOOK AT JAVA TOOLS FOR EMBEDDED SYSTEMS

Why Is Java Technology Interesting to the Embedded Market?

The embedded market's interest in Java technology is similar to the interest in Java technology on the desktop. Engineers and device manufacturers want to be able to develop quality code quickly and run the software predictably on different hardware without rewriting code. The "write once, run anywhere" promise of Java programming is very attractive in this marketplace because of the various RTOS (Real-Time Operating System) and hardware combinations. The ability to easily take existing code from one RTOS and run it on another is very appealing to vendors who must continually port their C/C++ and assembler code to each new device. However, as discussed later, using Java requires porting the VM (virtual machine) to the RTOS and the processor the engineer wants to use. Given the large diversity of processors in use in the embedded marketplace, this is a daunting prerequisite. However, once the investment is made, reuse of application code becomes easier.

The amount of function being implemented in software instead of hardware is increasing due to market pressures. Engineers need to adopt new programming technologies such as Java. They're excited about the Java language because it's a well-designed, object-oriented programming language. The concepts built into the language and the way it avoids many of the complexities and error-prone areas in C++ are attractive.

Marketplace Survey Experience

Our group started with a survey of the marketplace to see which VMs existed for which RTOSs. This proved a much harder task than anticipated. The information was gathered by attending conferences such as the Embedded Systems Conference and by searching the Web; we then proceeded to compile a matrix of available tools, VMs, RTOSs, and processors. We learned the following:

Claims Exceed Reality

The marketing brochures can be misleading. When we started this work in March 1999, the marketing brochures led us to believe that a VM is already ported to an RTOS. Further inspection reveals that it may not be available for another six or nine months. Though the marketing brochures are still not very informative, the embedded virtual machines are now becoming available.

Lack of Information

The information on many Web sites is incomplete. In many cases, the information on the Web is insufficient to determine what hardware is supported. For example, a Web site may state that it supports processors from Motorola but not have any more details. In all cases, the companies needed to be called directly to acquire enough information to be able to make a purchase decision.

Desktop Java developers are familiar with looking for a VM that supports the operating systems they choose for development. In the embedded environment this process is more complex.

Multitude of Processors

An embedded VM is written to a particular RTOS but compiled for a specific processor instruction set. Because the VM is actually an executable typically written in C, it must be compiled to a particular instruction set. For instance, the HP Chai VM supports the LynxOS operating system for the Motorola PowerPC processor, specifically the

“
The embedded
market's interest
in Java
technology is
similar to the
interest in Java
technology on
the desktop
”

MPC821 processor. It's not sufficient to ask which processor manufacturer (e.g., Motorola) the VM supports – you must be specific about which processor will support the instruction set the VM is compiled to.

Boards and RTOSs

An RTOS supports a specific board, not just the processor. The RTOS must be able to communicate with all the devices connected to the board. For example, the OS-9 operating system from Microware supports specific boards such as the Motorola MBX board. Running an RTOS on a particular board is achieved through a board-support package that's usually provided by the RTOS vendor. When you order the operating system and board, order the board-support package as well.

For an embedded developer these issues may not seem noteworthy. An embedded engineer typically starts with hardware specifications, such as the processor, and then seeks out the software. The typical Java developer usually decides on the software first, then seeks out the appropriate hardware. This was the approach we followed and learned that we needed to be very specific when gathering information.

Many Java Platforms

It's important to understand the profile or type of Java supported by the VM. There are different Java configurations available, such as PersonalJava and EmbeddedJava. When comparing VMs it's important that you understand what Java functionality is available in the VM.

- **PersonalJava:** Designed for network-connectable applications for devices, it can also run applets unlike EmbeddedJava. There are numerous implementations from RTOS vendors.
- **EmbeddedJava Application Environment:** For embedded devices with dedicated functionality and severely limited memory, it doesn't have a requirement for Web browsing or file systems. EmbeddedJava is intended to run on top of an RTOS. It's an application-driven subset because only the APIs required by the application are included.
- **JavaCard:** This API is used in embedded programmable chips that can be embedded in credit cards or other devices. Usage of smart cards is growing in Europe but hasn't penetrated the U.S. market yet. The JavaRing, showcased at JavaOne '98, is an example of support for the JavaCard.
- **Java2 Platform, MicroEdition (J2ME):** It was introduced at JavaOne '99 as a framework for Java for a variety of devices. A transition from PersonalJava and EmbeddedJava to the J2ME framework is planned. In J2ME there are two types of configurations: one for connected devices and the other for limited connected devices. A configuration consists of a VM and core libraries. One configuration is known as the CVM where the C stands for compact, connected, and consumer. This configuration is being developed as part of the Java Community Process, Java Specification Request - 0036. The other configuration is the K Virtual Machine (KVM), described below. Built on top of the configurations are profiles (classes) that are customized for the specific targeted market segment. The Personal Profile will be the next generation PersonalJava with profiles for TV, screen phone, auto, and handheld.
- **K Virtual Machine (KVM):** The KVM is the VM for the limited connected device configurations. It'll have profiles built to support wireless and handheld devices, and is available for the Palm. The KVM is targeted for a very small footprint and consumes little power. Additional development work on the KVM is in conjunction with the Java Community Process, Java Specification Request - 0030.

Many Proposed Solutions

There are a variety of solutions available to provide Java support in the embedded space. These solutions can be categorized by their approach to solving the challenges of running Java in an embedded environment or by the Java platform that's supported. In this article, tools from three different Java platforms are investigated.

- **IBM VisualAge MicroEdition on QNX Neutrino:** A subset of PersonalJava
- **Microware PersonalJava on OS-9:** A PersonalJava implementation
- **Sun KVM on PalmOS:** Part of the Java 2 MicroEdition platform

VisualAge MicroEdition

VisualAge MicroEdition includes a virtual machine and a fully integrated development environment to edit, compile, link, profile, and control the versions of your Java code. In addition to the embedded Java support, there's also a set of classes for serial I/O, voice, and user interfaces.

Virtual Machine Description

The VM, called J9, is based on the Sun JDK1.2 class libraries. You can use the J9 VM from the command line or make use of the integrated development environment that's included with VisualAge MicroEdition.

Tools

The integrated development environment has some similarities to IBM's VisualAge Java product. When using VisualAge MicroEdition, you can work in a stand-alone environment or in a team where each developer is connected to a server. The server contains a repository that allows developers to share code. Within the IDE you can group your code into projects, edit, compile, and debug on the host, or debug remotely.

The Remote Debugger provides the capability to debug embedded programs on the target virtual machine, either directly on the embedded target or simulated in the J9 Windows runtime environment. A productive feature is the ability to change the source code while debugging, then continue with the debugging session using the changes without restarting the debugger or program.

The Profiler allows you to profile your code that's executing on the target machine. In addition to the standard performance tuning, it can help solve threading problems and understand where garbage collection is occurring.

The Version Control doesn't follow the traditional check-in/checkout model in which code is locked while an engineer is working on it. In this product the engineers work in parallel. In the event that multiple engineers are working on the same class, the merge function enables them to merge the changes into the final version.

MicroView is the GUI framework that's about one-third the size of the AWT. It follows the model-view-controller architecture. The controller handles input events for the views and the model holds the state information. The model is observed by listeners who are notified of changes. As you'd expect, views can be nested.

The serial I/O classes are useful to communicate to other devices over RS232 serial ports. Use the `SerialPortConfiguration` class to set the configuration settings. There's a `SerialPort` class that's used with `SerialInputStream` and `SerialOutputStream` classes to read and write data.

The Voice classes consist of the `VoiceDispatcher` and the `Listener` interfaces that receive a voice event and handle it.



Building for the Target

When building your code, you can compile it into a .class file or .jar file. The J9 virtual machine allows you to store preprocessed classes in ROM. These classes are created by running the SmartLinker. To set the appropriate link options there's a GUI front-end to the SmartLinker tool. The output is a .jxe file and can be run on the target or Windows for simulation purposes. The J9 virtual machine still keeps a small portion of information in RAM for each class. Of the total runtime structures for a typical class, however, 75% can be stored in ROM with only 25% in RAM.

VisualAge class libraries are a subset of JDK1.2 for embedded applications. You have a choice of linking with one of three class libraries. The smallest library is `jclXtra` and is only 100K. It's a subset of the `jclCore` library and has no debug support. The `jclCore` library, which is about 300K, is used most often. The `jclCore` library has no security but does contain basic file I/O and networking support. The largest library is `jclMax`, which is about 2Mg. It includes most of the Java 2 APIs from the following packages: `java.io`, `java.lang`, `java.math`, `java.net`, `java.text` and `java.util`. It also includes Java 2 security.

Microware Personal Java for OS-9

PersonalJava for OS-9 includes the OS-9 operating system, the virtual machine, and a set of tools to help build the target.

Virtual Machine Description

PersonalJava for OS-9 is a Sun-licensed PersonalJava port that's been tuned for OS-9. The JDK version 1.1.6 is installed on the host for development. This PersonalJava implementation supports many optional features such as scrollbars and overlapping windows.

OS-9 has a modular architecture that enables dynamic loading of any OS-9 system or application module. This feature can be exercised to dynamically load Java programs.

In the architecture of OS-9 each process is protected from the other, so a bug in one process won't affect another. The Java thread scheduler and the OS-9 process scheduler are also independent. Therefore Java threads can be run independently from other processes executing real-time activities. PersonalJava for OS-9 contains a nonintrusive garbage collector that doesn't stop the entire system when it runs.

Tools

Along with the JVM are Java tools such as `JavaCodeCompact`, which allows a set of Java classes to be preloaded into RAM or converted into a ROMable shared library module.

A configuration wizard called `mwWizard` is available to help build system boot images. This is a GUI tool that allows you to select what should be included in the boot image and then creates the boot file. It can be very useful if you're iteratively rebuilding your boot image to make it smaller.

When writing Java programs, a developer is free to use any integrated development environment. `Hawk`, which is shipped with the product, is a C/C++ integrated development environment. It contains a syntax-sensitive editor, debugger, compiler, linker, profiler, project manager, and version control. From a Java perspective, `Hawk` is useful to dynamically load Java modules. The product contains two types of debuggers: `Microware Hawk` and `ROMBug`. The `Microware Hawk` debugger is a typical C source code debugger that also allows you to see the native assembler code. The `ROMBug` tool is used to debug system and user state programs. It runs in a supervisor state and takes control of the CPU when invoked.

The mwSoftStax component provides a C application programming interface to a variety of networking protocols. The LAN communications package contains the set of protocols.

Providing graphics capabilities are a graphics library with C APIs called mwMAUI (multimedia application user interface), a window manager called WinMgr, and a set of application framework classes.

An extensive set of online reference manuals in PDF format and Adobe Acrobat Reader with Search V3.0 are included.

Building for the Target

There are three ways to run PersonalJava in OS-9 when there's no hard drive on your target board. The first method prelinks the Java class files into an OS-9 module, which is then loaded into ROM or RAM. The second method involves including the classes.zip and the Java application code into a ModMan archive. The ModMan archive utility (mar) creates a single loadable file from a series of modules. The third method uses remote class loading.

The first method can be used to load the classes directly into ROM. To create the ROMized classes, the JavaCodeCompact tool is used. JavaCodeCompact creates the internal virtual machine data structures that represent the class in ROM. Since the JavaCodeCompact tool preloads the classes, information about the classes can be shared between them, resulting in a further savings in ROM. All symbolic references are resolved and the bytecode instructions referring to these symbols are rewritten to not perform symbolic lookups. PersonalJava for OS-9 ships JavaCodeCompact in the file called jcc.zip and a set of makefiles that invoke the JavaCodeCompact tool.

PersonalJava for OS-9 offers two versions of the Window Manager so if an application uses only basic window management functions, it can use a smaller library. One difference between the window managers is support for overlapping windows.

Sun KVM

The KVM is a new Java implementation developed at Sun Microsystems Laboratories, originally called the *Spotless System*. The KVM is targeted at providing a minimal set of functions for running on limited connected devices.

Virtual Machine Description

The KVM is currently available for download to run only on the Palm. It was also demonstrated on a Motorola pager at JavaOne '99. The system requirements for the KVM are 128K for the VM, libraries, and heap on the PalmOS.

The major design criteria was small size and portability. To achieve this goal, the garbage collector was redesigned to work with smaller heap sizes and the class library was reduced. AWT is replaced with a smaller GUI library. For example, instead of the Applet class, there's a Spotlet class. In some cases an entire class from the JDK is missing. Classes were removed if they were rarely used or created many short term objects. For example, there's no Float class. In other cases, though the class exists, some of the methods aren't supported. For example, some of the Math and String methods were removed. There's also a reduction in the number of distinct exceptions.

Tools

The only tool support is an emulator (POSE) that can be installed on the host machine. Once the emulator is installed, you can drop applications on it and see how they run before deploying them on a Palm.

Building for the Target

To develop a Palm application in Java, there are four steps:

1. Set your path and classpath.

```
set path=c:\jdk1.2.1\bin;
set classpath=
```

2. Compile your Java code using the KVM class library. It's recommended that your classpath point to the KVM classes.zip file instead of the desktop JDK as coded below using the bootclasspath option on the javac command. With this approach, if your code attempts to reference a class that's not in the KVM, a compiler error is generated instead of a runtime exception. The following statement assumes that the KVM was installed in the c:\kvm directory.

```
javac -O -bootclasspath c:\kvm\KVMDR4.1_bin\api\classes.zip HelloWorld.java
```

3. Make a Palm PRC file that contains your code. The MakePalmApp utility is located in the classes.zip file that's located in the tools directory. It takes a .class file(s) and converts it into a single .prc file that can be transferred to the Palm.

```
java -classpath c:\kvm\kvmDR4.1_bin\tools\classes.zip
palm.database.MakePalmApp HelloWorld
```

4. Transfer the PRC and KVM (if not already done) to the Palm. Start the HotSyncManager. Using the Palm Install tool add the application PRC file created in the previous step to the list of files to HotSync. If the KVM hasn't been transferred yet, add the KVM.prc and KVMClass-DB.pdb to the list. Perform the HotSync.

Another way to load Java code on the Palm is to merge the application .class files into the Palm database for the JDK classes using the MakePalmDB utility. This is convenient if you have classes that will be shared across applications. The first step is to download the kxfer utility and install it on the Palm. Then use the MakePalmDB as follows assuming that the HelloWorld.class file is in the current directory:

```
java -classpath c:\kvm\kvmDR4.1_bin\tools\classes.zip
palm.database.MakePalmDB HelloWorld
```

This creates a .pdb file that when downloaded to the Palm, adds your classes to the KVM. You then run your classes using the class manager. To do this select the KVM icon. From the list of classes in the KVM, select the one you want to run and press the Run button. The class needs to have a main function to run. You can see the results of stdout and stderr on the Palm. There's no way to delete a class once you have added it to the .pdb file.

As mentioned earlier, the KVM supports a more limited class library than PersonalJava. The GUI framework contains the basic classes such as Bitmap, Button, CheckBox, Dialog, RadioButton, Slider, and TextField. These are used within a Spotlet (instead of an Applet). The Spotlet class is used to interact with the user and it provides callbacks for pen and key events. It also provides methods for beaming data. There's no PrintStream class. To do simple output, use the printString method in the Graphics class.

TOOL	JAVA LEVEL	JAVA IDE	BOOT IMAGE BUILD TOOLS	.CLASS FILES STORED IN
IBM VAME	subset of Personal-Java 1.2	Yes	No	RAM or ROM
Microware Personal Java for OS-9	Personal Java built with JDK1.1.6	Yes, to load modules	Yes	RAM or ROM
Sun KVM	J2ME built with JDK1.2	No	No	RAM

TABLE 1 Tool Summary

Conclusion

As we surveyed the marketplace, we found that this is a very exciting and dynamic time to be doing Java development as the Java embedded market is just starting to emerge. Many RTOS companies are providing a Java Virtual Machine for their operating system, such as Microware PersonalJava for OS-9. The application development tools that are geared for Java development, not just C/C++, are typically following the release of the virtual machine and are starting to appear in the market. IBM VisualAge MicroEdition is an example of a development environment for the embedded programmer. As shown in the Tool Summary table (see Table 1), the tools on the market support different levels of the Java platform. Most tools allow the engineer to package the code to be loaded into RAM or ROM on the target.

The Sun Java Community Process and the recent J2ME announcements point to future changes in the embedded Java community. The changes to various JVM configurations with a set of profiles for their particular targeted use will have an effect on the solutions vendors provide. In fact, additional development for the KVM is available via the specification for the Connected Limited Device Configuration (JSR #30), which is accessible for public review on the JavaSoft Web site. There's also ongoing work to add real-time extensions into the Java Language Specification that will standardize solutions to some of the challenges faced in the embedded environment. ☛

References

1. Barr, M. (2000). "Developing Embedded Software in Java." Embedded Systems Conference.
2. Brenner, A. (1999). "K Virtual Machine Overview." Sun Microsystems, Inc.
3. Dibble, P. (1997). "Synergy: OS-9 and Java." Microware Systems Corporation. www.microware.com.
4. Grehan, R., Moote R., and Cyliax I. (1998). *Real-Time Programming:*

A Guide to 32-Bit Embedded Development. Addison-Wesley.

5. Real-Time Java: www.rty.org.
6. Reveaux, T. (1999). "Embedded Platforms for the Free Flow of Information." Strategic Platforms. *The Journal of Advanced Software Engineering*, Spring.
7. Sun Microsystems. (2000). "EmbeddedJava Technical Overview." www.javasoft.com/products/embeddedjava/overview.html.
8. Taivalaari A., Bush B., and Simon D. (1999). "The Spotless System: Implementing a Java System for the Palm Connected Organizer." Sun Microsystems, Inc.
9. Vokach-Brodsky B. (1999). "Java2 Platform, MicroEdition CDC Overview." Sun Microsystems, Inc.
10. Waters J. K. (1999). "Java Driving a Revolution in Embedded Systems." Strategic Platforms. *The Journal of Advanced Software Engineering*, Spring.

AUTHOR BIOS

Sherry Shavor is an advisory software engineer with IBM in Research Triangle Park, North Carolina. She received a BS in computer science from the State University of New York at Stony Brook.

Peter Haggard is a senior software engineer with IBM in Research Triangle Park, North Carolina and the author of the book, *Practical Java*, published by Addison-Wesley. He received a BS in computer science from Clarkson University.

Greg Bollella, a senior architect at the IBM Corporation, is lead engineer of The Real-Time for Java Expert Group. He holds a PhD in computer science from the University of North Carolina at Chapel Hill.

sshavor@us.ibm.com

haggard@us.ibm.com

bollella@us.ibm.com

Short List

Secrets of Java Serialization

WRITTEN BY
GENE CALLAHAN &
ROB DODSON



The serializable interface doesn't contain any code or data; it's a marker interface. If a hierarchy of classes is to be serialized, each class in the hierarchy must implement the serializable interface. All objects that are in an object hierarchy, or "Web of objects," at runtime will be serialized. If any one of them doesn't implement serializable, an exception will be thrown.

Serialization works across platforms – an object serialized on Solaris can be read on Windows. This is a key component in making Java's write once, run anywhere philosophy real, as well as the core requirement that Java programs be able to effectively and easily communicate with each other.

The default serialization capabilities often do the job. When that's not the case, Java allows progressive customization of the process at the class level. A class can contain its own `readObject()` and `writeObject()` methods to add data to the stream, set variables to special values, and more. We've found that we use `readObject()` frequently to reinitialize fields that it doesn't make sense to serialize, such as a database connection handle. We use `writeObject()` a lot less.

For those who really need a custom solution, Java provides the externalizable interface, which allows (in fact, forces) the class that implements it to take complete control of reading and writing instances of itself to the I/O stream. In over a year of using serialization every day, we've never had to resort to this level of customization. Still, it's nice to know that it's there.

Serialization in Java is an operation in which an object's internal state is translated into a stream of bytes. This binary stream or image of the object is created in an operating system-neutral network byte order. The image can be written to a disk, stored in memory, or sent over a network to a different operating system. This amazing feat requires little or no work on the part of the programmer. Just implement the serializable interface, which contains no methods, and call the `writeObject()` method on your object, and it's serialized! You can serialize an object to or from any I/O device that Java supports.

What Are Some Uses for Serialization?

Network

Serialization is much easier than writing message formats for each type of message that is to be passed between a client/server pair.

Just send the common objects back and forth and you're done. Designing messaging protocols is notoriously finicky. Once again Java saves the day and makes a previously arduous task a simple matter of a single method call.

Serialization over networks is the basis of Sun's communications infrastructure: RMI uses it, as well as the many communications subsystems in Jini such as JNDI.

File I/O

As we stated before, an object may be serialized to any Java I/O byte stream class. So it's trivial to serialize an object or a whole hierarchy of them to a disk file. We use this feature of the language for two reasons:

1. *Saving an application program's state for the next time it's started.*
2. *Caching objects on a disk before they're needed.* These objects are complex hierarchies that are created from database access and extensive algorithmic processing, and thus must be created ahead of time.

RDBMS

Another use of serialization is to store complete objects as BLOBs in a relational database. This can provide a way to use traditional databases with an object-oriented programming language

What is serialization?

without needing to move to a true object-oriented database.

Tips and Traps

Trap 1: Hidden Caching

We found our first trap in a client/server setting. We wanted to resend objects when their values changed. But we saw that although the object had been sent again, it still had its original value. Much to our surprise we discovered that the default behavior for Java serialization was to serialize any unique object (as determined by a comparison of the memory address of the object to be serialized with that of all objects previously serialized) just once. The serialized form is cached and then sent again when requested. This design decision helps speed up serialization for applications that are just passing objects as messages, not for their internal values. However, it's not obvious to beginning users that this caching is happening. Each time you try to serialize the object, you'll get that first cached instance again. If you want the serialized object to reflect changes in the "source" object, call the `reset()` method.

Trap 2: Versioning

If you attempt to send a serialized object to a running program that knows only about an earlier version of the class, an exception will be thrown. This happens when a class is compiled because the Java compiler creates a version stamp in the class. When the internal structure of the classes changes, the version stamp is changed. So when a serial-

ized object is received by a program, the version in the object must match the class version stamp the program knows about. If they don't match, Java will throw an exception to prevent even worse things from happening, such as old code trying to read a new object.

You can take over version stamp creation yourself, but watch carefully for versions that really aren't compatible (such as the addition or deletion of a new class member). If you do want to take control of versioning of a class yourself, you must declare the following in the class's source code:

```
Static final long serialVersionUID
= 12; // 12 is just an example!
```

Whenever you make changes that would prevent compatible serialization, bump up the version number.

Rather than trying to keep track of this ourselves, we let Java do the work for us through the class `java.io.ObjectStreamClass`. We prevent the “incompatible version” problem by sharing JAR files between programs that send serialized objects back and forth. The common JAR files contain all the shared classes. This prevents two different programs from trying to use different versions of the same class.

Sun also changes the serialization format occasionally. This is something to watch for if you're generating a serialized object store with some persistence, for instance, a disk cache that will remain around for some time. Fortunately, Sun seems intent on incorporating ways to work around the problem when it does make these changes. Sun posted the following on the Java Web site (www.java.sun.com/) with the release of version 1.2:

It was necessary to make a change to the serialization stream format in JDKTM 1.2 that isn't backwards compatible to all minor releases of JDKTM.

1.1. To provide for cases where backwards [sic] compatibility is required, a capability has been added to indicate what `PROTOCOL_VERSION` to use when writing a serialization stream. The method `ObjectOutputStream.useProtocolVersion` takes as a parameter the protocol version to use to write the serialization stream.

Tip 1: Handling Static Variables

Java classes often hold some globally relevant value in a static class variable. We won't enter into the long history of the debate over the propriety of global variables – let's just say that programmers continue to find them useful and the alternatives suggested by purists aren't always practical.

For static variables that are initialized when declared, serialization doesn't present any special problems. The first time the class is used, the variable in question will be set to the correct value.

Some statics can't be initialized this way. They may, for instance, be set by a human during the running time of the program. Let's say we have a static variable that turns on debugging output in a class. This variable can be set on a server by sending it some message, perhaps from a monitor program. We'll also imagine that when the server gets this message, the operator wants debugging turned on in all subsequent uses of the class in the clients that are connected to that server.

“
Instead of writing
ugly and
hard-to-maintain
clone methods,
simply
serialize the
object to
memory, read it
back to a new
reference, and
you have a new
deep copy
”

The programmer is now faced with a difficulty. When the class in question arrives at the client, the static variable's value doesn't come with it. However, it contains the default static state that's set when the class's no-argument constructor is called by `writeObject()`. How can the client programs receive the new correct value?

The programmer could create another message type and transmit that to the client; however, this requires a proliferation of message types, marring the simplicity that the use of serialization can achieve in messaging. The solution we've come up with is for the class that needs the static transmitted to include a “static transporter” inner class. This class knows about all the static variables in its outer class that must be set. It contains a

member variable for each static variable that must be serialized. `StaticTransporter` copies the statics into its member variables in the `writeObject()` method of the class. The `readObject()` method “unwraps” this bundle and transmits the server's settings for the static variables to the client. Since it's an inner class, it'll be able to write to the outer class's static variables, regardless of the level of privacy with which they were declared.

Tip 2: Easy Cloning

Serialization provides a simple way to clone. Instead of writing ugly and hard-to-maintain clone methods, simply serialize the object to memory, read it back to a new reference, and you have a new deep copy. The deep-copy idea is important. When a shallow copy is performed (the default behavior of Java cloning), only references to data members are copied. If an object of type `Foo` holds a reference to a `String s`, when a `Foo` is cloned, both the original and the new copy will point to the same copy of `s`. Sometimes this is fine, but other times you need a deep copy.

In a deep copy the new object will get new copies of its data members, not just new pointers to the same data members. Most of the time when you want this behavior, you want it recursively so that the members' members are deep copied as well. A major problem with clone methods is the difficult code that's required for all the deep copying a class might require. To solve this problem we wrote a class called *Cloner* that removed the need to write clone methods at all. *Cloner* is small enough so it can be included here in its entirety:

```
public class Cloner(){
    public Cloner() { }
    public static Object
clone(Object o) throws Exception {
    ByteArrayOutputStream b = new
ByteArrayOutputStream();
    ObjectOutputStream out = new
ObjectOutputStream(b);
    out.writeObject(o);
    out.close();
    ByteArrayInputStream bi=new
ByteArrayInputStream(b.toByteArray());
    ObjectInputStream in = new
ObjectInputStream(bi);
    Object no = in.readObject();
    return no;
}
}
```

To use this class is easy:

```
Foo foo = new Foo();
Foo bar = (Foo)Cloner.clone(foo);
```


Now “bar” is a completely new deep copy of Foo!

As Bruce Eckel points out in his excellent book *Thinking in Java*, it's an order of magnitude slower to clone this way. However, in situations where you need to do a deep copy and are more worried about development time than running time, this is a good alternative. And since objects aren't cloneable by default, you need access to their source to make them so. You can subclass an object just to make it serializable, but this only works if you're the one constructing it. If you need to deep copy a Web of objects, some of which are created inside libraries that you don't have the source for, serialization may be your only alternative.

AUTHOR BIOS

Gene Callahan, president of St. George Technologies, designs and implements Internet projects. He has written articles for several national and international industry publications.

Rob Dodson is a software developer who writes options-trading software in Java and C++ for OTA Limited Partnership. Previous projects include weather analysis software, tactical programs for Navy submarines, and code for electronic shelf labels.

Tip 3: Transient

Make liberal use of the transient keyword to trim down the size of your serialized objects. Mark any class elements that don't need to be passed between programs as transient. Some of these elements, such as file handles, are useless when passed to another program. Other elements are needed, but can be re-created from other data members. This can be done by writing your own readObject() method.

For example, perhaps you store some objects in a linked list, but also keep them in a hashtable for quick

“
You can subclass an object just to make it serializable, but this only works if you're the one constructing it.”

lookup by value. Since the hashtable can easily be re-created from the list, you can mark it as transient. If the table is large, re-creating it can be much faster than serializing, transmitting, and deserializing it.

The use of transient can also be important for security considerations. If a class contains data members that hold information that shouldn't be made public, such as a clear text version of a password or employees' salaries, a malicious program might be able to read them from the serialized byte stream. Declaring those members transient will prevent them from ever being written to serialized output. (Java also allows for the encryption of byte streams if further security measures are needed.)

Summary

Serialization is an important and useful addition to the Java language. However, before you make use of it, it's important to understand the pitfalls of the technique and know how to turn its strengths to your advantage. Sun has made it the foundation of Java's communications infrastructure. You can make it the foundation of yours as well. ☛

gcallah@erols.com

robododson@erols.com



The employment portal for IT professionals

<http://careers.sys-con.com>

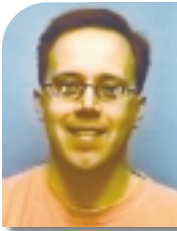
Dive into your next IT Job with confidence!

SYS-CON MEDIA CAREER CENTER

Directly access the best companies in the nation. If you are an IT professional and are curious about the job market, demand privacy, and don't want to waste time, you have found the right site.

Persistence Frameworks

Save time and money and improve your database apps



WRITTEN BY
MIKE JASNOWSKI

Most application architectures are organized into tiers. Presentation, business logic, and data combine to form a complete solution from end to end. The data tier is where your application gets and stores data that's used throughout the application. It could be accessing relational- or object-oriented databases or native file stores or connecting to a mainframe to obtain its data.

Application architectures also need to be designed to minimize impact on various components of the application. Another common goal among many database applications is to take unwieldy rows and columns and turn them into manageable application objects.

Without a good design, database applications have a high potential for problems. Changing table structures and column names can wreak havoc on an application. This article focuses on use of a persistence framework to access application data – TOPLink for Java – in a functioning Web-based application. I'll demonstrate how easy it is to create a database application using a persistence framework, the time-saving over hand-coding SQL, and the supporting code to turn it into a Java object. I'll also point out advantages a persistence framework provides in areas other than removing SQL code from your application code. (The listings for this article can be found on the *JDJ* Web site.)

The toolkit discussed in this article consists of:

- JDK1.2
- IBM DB2 Universal Database Personal Edition (sample database)
- DB2 JDBC driver from IBM
- TOPLink for Java 2.5
- Allaire JRun 3.0

See the "Links" section at the end of this article for information on obtaining evaluation copies of some of the software used. My example application requires the use of TOPLink for Java, the other software may be replaced by your preferred choice. (While I discuss the features of this product in some detail, this isn't a product review.)

Persistence

The storage of data used in an application is sometimes referred to as *persistence*. Data that's used is persisted between uses of the application by an individual user. Things such as account information, user preferences, or application data can be stored and later retrieved by a user. There's more than one type of persistence. Object-relational persistence is one of the more common methods of persisting data. Object-relational persistence involves the mapping of attributes of an object to the rows and columns of a relational database table (see Figure 1).

Before persistence frameworks, some applications may have been written in a fashion similar to that shown in Listing 1. However, as applications increased in complexity, application designs such as the one in Listing 1 would become

unmanageable. An increase in the complexity of the data model could also break it. By providing a framework for managing the access not only to your data but to the process of turning that data into objects, you can reap benefits in both cost and time of development.

TOPLink for Java

TOPLink for Java is an object-relational persistence framework. It's available for other languages but its use in this article will focus on Java. Features include:

- Handling of different types of mappings, one-to-many, many-to-many
- Caching of retrieved objects
- Expression classes for constructing queries
- Named queries and query keys for abstracting queries from field names in your table

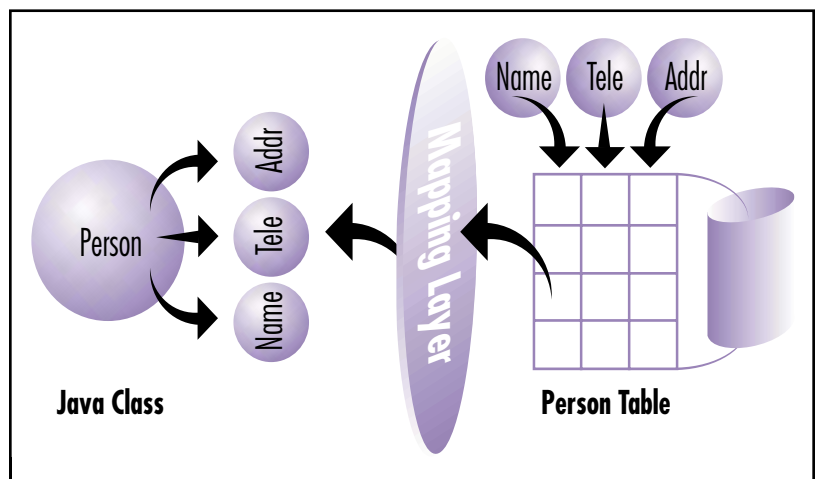


FIGURE 1 Mapping layer

- Descriptors that are external files used to describe class-to-table relationships
- A UnitOfWork, which encompasses changes to table(s) inside a transaction as well as rolling back changes (should problems occur), even across multiple tables
- Indirection, which causes reading of related objects to be deferred until they're accessed

TOPLink allows you to map your objects and attributes to the rows and columns of your relational database tables. You can perform this mapping at runtime using the TOPLink API, or at design time using the TOPLink Builder.

Sample Application

Our sample application will be a simple Web-based phonebook. It will present the user with a primary display of a phonebook. Information will be stored for the name, address, and telephone number. Functionally it provides some basic features that will exercise the persistence framework such as reading and adding new records. Following are the steps we'll take to build this example:

1. Create the database tables.
2. Create our business objects.
3. Create our mappings using TOPLink Builder.
4. Create our application logic and user interface.
5. Run our application.

Create the Tables

The tables used during this application are composed of persons and phonebooks. You can run the following SQL statements to create the necessary tables and insert some dummy data into the phonebook table.

```
CREATE TABLE PERSONS(
  AREACODE CHAR(16) NOT NULL,
  NAME CHAR(40) NOT NULL,
  ADDRESS CHAR(60)
  NOT NULL,
  TELEPHONE CHAR(15),
  PRIMARY
  KEY(AREACODE,NAME,ADDRESS)
)

CREATE TABLE PHONEBOOK(
  AREACODE CHAR(16) NOT NULL,
  PRIMARY KEY(AREACODE)
)

INSERT INTO PHONEBOOK (AREACODE) VALUES('12345')
INSERT INTO PHONEBOOK (AREACODE) VALUES('67890')
```

Create the Business Objects

The business objects used in this application are composed of a class called *person* and one called *phonebook*. Phonebook holds a vector of person objects (see Listings 2 and 3).

Create TOPLink Mappings

Once we've created our objects and database tables, we can start creating our TOPLink mappings using the TOPLink Builder. Essentially, the steps we need to take are:

1. Specify the database login information including the JDBC driver to be used.
2. Import our classes into the TOPLink Builder.
3. Import our database table into the TOPLink Builder.
4. Map the classes to the tables.
5. Save our project.

I won't go into explicit detail on how to accomplish the mappings in the builder, as the TOPLink documentation should cover this. When you're finished, your TOPLink builder project should look similar to Figure 2.

will fail with an exception indicating a similar error.

Create the Application Logic and User Interface

Now that we have our objects mapped, we need to create the application logic and the user interface. The UI for this application will allow us to:

- View all area codes in our phonebook.
- View the phonebook for a specific area code.
- Add a new entry to the phonebook for a specific area code.

I've tried to keep the application as simple as possible, although applications significantly more complex are possible with the persistence framework used. This application is broken into an index page, a ViewPhoneBookServlet, and a ViewAreaCodeServlet. This code isn't intended to be production quality. It's designed simply to illustrate the use of a persistence framework in your application. We'll also create a specific class to handle all our TOPLink start-up requirements (see Listing 4). TOPLinkSession makes the connection to our database, providing a session handle through which we can interact with the

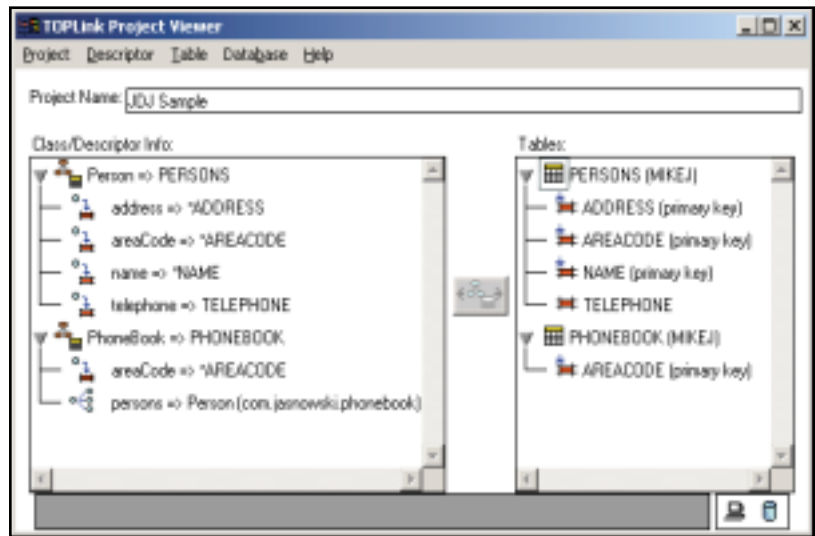


FIGURE 2 TOPLink Project Viewer

When developing with TOPLink, there are two items you should note:

1. TOPLink requires each table or view that's mapped to have a primary key.
2. Each primary key must be mapped to an attribute in the corresponding Java class.

If you're using the TOPLink Builder to perform your mappings, it will indicate which keys are unmapped as well as which tables don't have primary keys specified. If you're not using the TOPLink Builder, when your application runs, it

persistence framework. The class takes care of loading the appropriate JDBC driver and logging into the database. Ideally, the TOPLinkSession should be a single instance, but in this application, each servlet that requires database access creates an instance. See Listings 5–10 for these files:

- *ViewPhoneBookServlet.java*: Displays a list of area codes from which to select
- *ViewAreaCodeServlet.java*: Displays a list of persons for a specified area code

- **AddNewPersonServlet.java:** Displays a page that allows you to add persons to an area code
- **Viewareacode.jsp:** Called by ViewAreaCodeServlet to display persons for an area code
- **Addnew.jsp:** Called by AddNewPersonServlet to add a new person
- **Phonebooks.jsp:** Called by ViewPhoneBookServlet to display area codes

I'll focus on two primary areas of the code, reading and writing to the database, and point them out in the code.

Reading from the Database

The application uses objects and methods of the persistence framework to read data from the database. In ViewPhoneBookServlet (see Listing 5), you'll notice the following line:

```
Vector pBooks = tlSession.getSession().readAllObjects(PhoneBook.class);
```

This line of code is all that's required to read data from the phonebook table and assemble it into a vector of PhoneBook objects. It calls the readAllObjects method of the DatabaseSession object contained in our TOPLinkSession class. The DatabaseSession method readAllObjects is just one of the many methods TOPLink provides to read data from the database. The SQL output from TOPLink for the following sequence of code would look something like this:

```
session(-335903629):
thread(1730053239): connection(-365525901): SELECT ADDRESS, AREACODE, NAME, TELEPHONE FROM PERSONS WHERE (AREACODE = '99999')
```

Writing to the Database

The application also utilizes objects and methods of the persistence framework to write data to the database. In AddNewPersonServlet (see Listing 7), you'll notice the following block of code:

```
1. UnitOfWork uow = tlSession.getSession().acquireUnitOfWork();
2. PhoneBook pBookClone = (PhoneBook)uow.registerObject(pBook);
3. Person personClone = (Person)uow.registerObject(newPerson);
4. personClone.setName(req.getParameter("NAME"));
5. personClone.setAddress(req.getParameter("ADDRESS"));
6. personClone.setTelephoneNumber(req.getParameter("TELEPHONE"));
```

```
7. personClone.setAreaCode(req.getParameter("AREACODE"));
8. pBookClone.addPerson(personClone);
uow.commit();
```

Writing requires somewhat more code, but is equally as straightforward. Line 1 acquires a UnitOfWork from the persistence framework. Lines 2 and 3 register the PhoneBook and Person objects we're about to use in our update. Lines 4–8 set the values of our new objects. Line 9 calls the commit method of the UnitOfWork object created in Line 1, and our update takes place, wrapped in a transaction. The SQL output from TOPLink for the following sequence of code would look something like this:

```
session(727882650):
thread(1391369113): acquire unit of work
session(1868471198):
thread(1391369113): begin unit of work commit
session(727882650):
thread(1391369113): connection(729979802): begin transaction
session(1868471198):
thread(1391369113): connection(729979802): INSERT INTO PERSONS (ADDRESS, AREACODE, NAME, TELEPHONE) VALUES ('123 Main St', '99999', 'AnyBody', '405-345-5678')
session(727882650):
thread(1391369113): connection(729979802): commit transaction
session(1868471198):
thread(1391369113): end unit of work commit
session(1868471198):
thread(1391369113): release unit of work
```

Run the Application

Once you've compiled all your classes and placed them in the appropriate subdirectories, start your browser and point it at <http://localhost:8100/servlet/ViewPhoneBookServlet>, the starting point of our application. You'll be presented with a list of area codes. Select an area code and press the GO button.

You'll see a page with some headings and an Add button. Press Add and fill out the form. When finished, press Save and you'll be redirected back to see your update. Open up the DB2 Command Center and run the following query, SELECT * FROM PERSONS.

You can see that the new person exists in the database, all without coding a single line of SQL! However, if you're interested in seeing what TOPLink is

doing, we can turn on logging and log any SQL to an external file. The following code snippet demonstrates this; it can be added to the TOPLinkSession class.

```
session.logMessages();
try{
    session.setLog(new OutputStreamWriter(new FileOutputStream("C:\\debugsql.log")));
}catch(Exception ex){System.out.println(ex);}
```

I'd also like to point out that when retrieving a phonebook, the persistence layer retrieved all persons associated with that phonebook using the AREA-CODE key.

Advantages/Disadvantages

Using a persistence framework provides the following advantages:

- Caching of retrieved objects
- Separation of SQL from application logic
- Separation of the data access layer from application
- Caching and indirection can provide performance benefits when accessing application data.
- Simplified access to complex database structures

Disadvantages are:

- Learning curve
- May require changes in your data model and/or current code

I'd also like you to compare Listings 1 and 5; you'll no doubt notice the difference. Regardless of the disadvantages, an investment in using a persistence framework can yield longer-term benefits as your applications increase in complexity and size.

Links

The following links can be used to obtain free evaluation copies of the software referred to here, as well as some of the other software used. Additionally, TOPLink will function with JDK1.1. The TOPLink for Java evaluation requires a temporary license key, which is emailed to you. Good luck!

1. *TOPLink for Java – The Object People/WebGain:* www.objectpeople.com
2. *JRun 3.0 – Allaire Corporation:* www.allaire.com
3. *Java 2 Standard Edition:* <http://java.sun.com>

boopan@msn.com

AUTHOR BIO
Mike Jasnowski, a Sun-certified Java programmer, has over 17 years of programming experience, and over four years with Java. He's a senior software engineer with eXcelon Corporation in Burlington, Massachusetts.


TogetherSoft Ships Together Control Center 4.1

(San Jose, CA / Raleigh, NC) – Together Control Center 4.1 from TogetherSoft Corporation is now available. It has a new XML structure editor, improved compile/make/run/debug actions, enhanced version-control options, improved forward and reverse engineering of database schemas, new visual-inspector customizer, and new WLE CORBA Expert for working with BEA WebLogic Enterprise.  www.togethersoft.com


Inprise Launches JBuilder 4.0

(Scotts Valley, CA) – Inprise/Borland is shipping JBuilder 4, the next generation of its award-winning visual development tool for Java. JBuilder 4 is the only enterprise tool set that supports EJB 1.1-compliant development on Windows, Linux, and Solaris platforms. New features include rapid J2EE application development, visual creation of Enterprise JavaBeans, integrated team development with CVS version control repository, Inprise Application Server 4.1 development license, WebLogic development support, and source code debugging of JSPs.  www.inprise.com

PointBase Licenses Java Database to Metrowerks


(Mountain View, CA) – Metrowerks, a Motorola Company, has licensed the right to include PointBase's industry-leading database in Metrowerks' CodeWarrior IDE. The agreement gives Metrowerks an embedded pure Java database, helping boost the productivity of Java developers writing database applications for mobile information devices.  www.metrowerks.com
www.pointbase.com

QA-Systems Introduces QStudio Java


(The Netherlands) – QStudio Java from QA-Systems offers the possibility of automating quality analysis and control for applications written in Java. A trial version of the product can be downloaded from the Web site.  www.qa-systems.com

Free Software Developer Tools from Espial

(Ottawa, Canada) – Espial is dedicating free developer versions of its Java-based software and development tools to Java developers on devicetop.com, the first portal for smart device developers.

To motivate Java developers to learn and create smart device applications, devicetop.com is putting a BMW Z3 roadster up for grabs. Devicetop.com will award the car to the developer who uploads the best Java technology-based smart device application to the devicetop.com portal. The contest finalists will be determined by a vote of the members of the devicetop.com portal.  www.devicetop.com
www.espial.com

RSW Software's e-TEST Suite and Bean-Test Standardized by Consulting Firm

(Waltham, MA) – Concrete Incorporated has standardized on RSW Software's Web application testing technologies e-TEST suite and Bean-test (formerly EJB-test). A high-end Internet consulting firm, Concrete utilizes RSW's testing solutions to verify and monitor the functionality and scalability of its client's Web applications.  www.concreteinc.com
www.rswsoftware.com

Rational Software Recognizes Ensemble Systems

(Richmond, BC) – Ensemble Systems was presented with the Rational Software Customer Choice Award at Rational's annual

WebGain Announces TopLink for IBM WebSphere App Server

(Santa Clara, CA) – WebGain Inc. announces the latest version of its award-winning Java object-relational database mapping technology with support for IBM's WebSphere Application Server, Advanced Edition v.3.5. 


TopLink features full inter-bean relationships, caching options, a powerful querying system, a flexible non-intrusive mapping mechanism and object-level transactions. www.webgain.com



al Rational Users Conference, recognizing Ensemble as the partner receiving the most votes in a recent Rational customer satisfaction survey. The award recognizes Ensemble as a leading choice for products and services complementary to Rational's products, including the market-leading Rational Rose visual modeling tool. www.ensemble-systems.com


Versant Offers enJin Early Adopter Release Program

(Fremont, CA) – Versant Corporation's enJin Early Adopter Release (EAR) program to help customers speed performance and enhance the scalability of their application server environments is now available.


This is the industry's first comprehensive middle-tier infrastructure platform to integrate with all EJB-compliant application servers and provide seamless synchronization with the back-end enterprise database. The program offers a complete package including one developer's license of the EAR 

software, three days of enJin training, and five days of enJin consulting. www.versant.com/enjinearlyrelease

Progress Launches SonicMQ 3.0

(Bedford, MA) – Progress Software Corporation has introduced Progress SonicMQ 3.0, the next generation of its e-business messaging infrastructure. SonicMQ 3.0 offers a dynamic routing architecture that  allows enterprises to participate in a global e-business exchange through a single point of entry. www.progress.com

Flashline Component Manager Ships with JBuilder 4

(Cleveland, OH) – Flashline Component Manager, a Web-enabling IDE add-in, is now included with Borland JBuilder 4.  Together, Component Manager and Inprise/Borland's leading Java Internet application development environment create a robust resource that speeds component-based software development. www.flashline.com

IBM VisualAge Micro Edition Gains Acceptance as Key Part of Embedded Auto Solutions

(City, ST) – IBM's VisualAge Micro Edition embedded Java 2 technology is fast gaining broad acceptance and is now being used by PSA Peugeot Citroën, Motorola, Daimler-Chrysler, and Intel in projects focused on the future of vehicle-based embedded computing. VisualAge Micro Edition is

being selected because it gives engineers and developers the tools and run-time technology they need now for building the next generation of automotive software technology.  www.ibm.com

Java Developer's Journal Becomes Fastest Growing Technology Title on Newsstands

(Montvale, NJ) – **SYS-CON Media, Inc.** (www.sys-con.com), headquartered in Montvale, New Jersey, has announced that the world's leading Java resource, *Java Developer's Journal*, is set to become the fastest-growing technology magazine on U.S. newsstands and throughout the English-speaking world.

"We have developed a progressive growth program for selected magazine titles that show future strong single-copy sales potential, and *Java Developer's Journal* is one of several titles we have chosen for that program," said Robert Castardi, president of Curtis Circulation Company.

"Single-copy sales of *Java Developer's Journal* in the United States are up an impressive 47% for the first six months of 2000."

"We're committed to bringing the high-quality editorial content of *JDJ* to Java professionals around the globe," said Fuat Kircaali, president and CEO of **SYS-CON Media** and publisher of *Java Developer's Journal*. "We're very pleased to see an impressive sales growth for *JDJ* on the newsstands as well as our subscriber base since *JDJ*'s first issue five years ago."

During the first six months of 2000, Curtis distributed 96,393 copies of *Java Developer's Journal* to specialty sales outlets to sell 68,291 copies – an efficiency of 70.8% – a success ratio not seen in any other technology title for the same period. Specialty store sales such as Barnes & Noble and Borders Bookstores showed an increase of 152.6% as compared to the same period of 1999. Curtis also distributes *JDJ* via its wholesalers and to independent bookstores.

Java Developer's Journal's BPA audited circulation has consistently exceeded the combined circulation of all other Java-related magazines for the past five years.

JDJ's worldwide newsstand distribution is managed by Curtis Circulation Company, the largest magazine distributor in the world. ●

SYS-CON Media Debuts SYS-CON Television at CTIA Wireless IT

(Montvale, NJ) – In October **SYS-CON Media, Inc.**, introduced **SYS-CON Television** at the CTIA Wireless IT Conference and Expo held in Santa Clara, California. **SYS-CON TV**, the premier source of Internet technology news and commentary, was Webcast live at www.sys-con.com.



During its first day of broadcasting, **SYS-CON Television** featured live interviews from the show floor with the movers and shakers of the hypergrowth wireless industry. "We've brought the excitement of live television to the CTIA show floor, and targeted this service with maximum effectiveness through the Webcast medium," noted Fuat Kircaali, founder and CEO of **SYS-CON Media**. "It was truly a great day for technology and for our company. **SYS-CON** is very pleased to add **SYS-CON Television** to our stable of Internet technology products and services." ●

SYS-CON Debuts Wireless Business & Technology Magazine at CTIA Wireless IT

(Montvale, NJ) – **SYS-CON Media, Inc.**, introduced the preview issue of its newest title, *Wireless Business & Technology* (www.wireless-magazine.com) at the CTIA Wireless IT Conference and Expo held in Santa Clara, California.



"Building on the remarkable success of **SYS-CON's** existing stable of technology portals and accompanying print magazines, *Wireless Business & Technology*

KL Group Becomes Sitraka and Charts New Enterprise Strategy

(Toronto, ON) – **KL Group**, a leading provider of advanced Java development, deployment, and management solutions, has announced the creation of **Sitraka**, an organization dedicated to providing reliable, innovative software and solutions that help enterprises leverage the power of the Internet. Under the new name, the company has formed two divisions: **Sitraka Software** and **Sitraka Mobility**. Each division will operate as a distinct organization with its own president, mission, and vision.

The **Sitraka Software** division will focus exclusively on software solutions that help customers develop, deploy, and manage Java-based Internet applications. It will continue to support and expand **KL Group's** products, including the award-winning **JProbe** and **JClass** product lines. **Sitraka Software** will develop and bring to market new breakthrough IT solutions such as the recently announced **DeployDirector** Java deployment product. In a separate news release, **Sitraka Software** announced the appointment of the division's new president, **Larry Humphries**.

"Java is the language, the environment and the future of the Internet," said **Humphries**. "Sitraka Software will be the world leader in helping customers build business-critical Java-based Internet applications. We are sharpening our focus and dedicating additional resources to the full spectrum of Java development, deployment, and management. Customers will benefit from having a partner that is entirely focused on this single mission."

features seasoned gurus as well as fresh new voices in the unwired world," said **Jeremy Geelan**, publisher of *WBT*. "The magazine features a rich mix of industry insight and analysis, together with real-world applications from which wireless devel-

The **Sitraka Mobility** division will be a leading provider of reliable, innovative software solutions and services that help enterprises gain competitive advantage in a mobile wireless world by leveraging emerging wireless technology. **Sitraka Mobility** inherits **KL Group's** world-class reputation for quality and innovation, leadership in emerging markets, award-winning products, and superior customer support. **Sitraka Mobility** also announced the first two products in its **Mobile Enterprise Architecture**, **ActionableAlerts** and **RemoteControl**.



Greg Kiessling, CEO of **Sitraka**, explaining the new vision of **Sitraka Software** (formerly **KLGroup, Inc.**) to **Jeremy Geelan** of **SYS-CON Radio** at the **CTIA Wireless IT Conference**.

"Sitraka's history as **KL Group** is one of proven leadership in emerging markets," said **Greg Kiessling**, CEO of **Sitraka**. "Both of our divisions will be leaders in their markets, and both will share the core values and award-winning best practices that made **KL Group** successful." Customers have rewarded **KL Group** with 11 consecutive years of profitable, double- and triple-digit revenue growth. In the past year alone **KL Group's** products received 11 awards, including six *JDJ* Readers' Choice Awards. www.sitraka.com ●

opers can learn and profit. *WBT* lifts the veil off mobile commerce and the emerging galaxy of unwired solutions that are becoming available for management, education, and communication in the mobile information society." ●

Create XML Based Web Applications Using IBM VisualAge for Java

XML and Java tools make development and debugging easy

WRITTEN BY
LUC CHAMBERLAND
AND ARTHUR RYMAN



Beyond the battle being waged on the first tier, the client, it's important to understand the drive toward scalable, platform-neutral technologies on the middle and back-end tiers. In particular, Java and XML set the stage for the growth of B2B Web marketplaces, where participants in supply chain management (SCM) and enterprise resource planning (ERP) will realize enormous gains in speed and access to data.

In this article we'll look at a simple application that uses Java and XML to manage data: the Conference Survey Web page. To develop complex Web pages, particularly pages based on a server-side programming model, programmer productivity is strongly affected by the tools used. We'll show how the IBM VisualAge for Java suite of tools can be used to develop, build, and test this page.

Conference Survey Web Page

At the end of the fictitious e-Business 2000 conference, participants are asked



FIGURE 1 Conference survey Web page

since the mid-'90s we've seen the quality of Web programming paradigms mature at an astonishing rate: from static pages with animation, CGI-based programs, and JDBC connectivity to back-end relational databases and servlets processing requests on application servers. We commonly hear about Web pages being more interactive, likely using HTML forms, JavaScript, or Java applets.

to fill in a survey to rate the various sessions and to provide comments and suggestions for future use. To ensure attendee responses aren't lost and don't need to be scanned or typed into a system, a Web page has been set up (see Figure 1).

Behind this HTML form is a three-tiered Web application (see Figure 2). The essential pieces of the application can be grouped as follows:

- **Tier 1:** HTML form, JSP pages
- **Tier 2:** Java servlet, JavaBeans, XML parser, all running under an application server
- **Tier 3:** XML files

In our Conference Survey application we opted to use HTML on the client Web browser instead of XML for two reasons:

1. HTML forms are prolific and drop-dead easy to implement. Simple forms support a thin-client model, in contrast to Java applets, which are too

cumbersome as a data transfer mechanism.

2. At this writing XML support in popular Web browsers is inconsistent. You don't want to find that your application isn't accessible because of the foibles of a browser.

The use of JSP pages allows us to dynamically build the HTML pages that are sent back to the user and to maintain a clear separation between presentation and business logic.

Tier 2, the middle tier, represents those parts of the application that run on an application server (e.g., IBM WebSphere Application Server). The servlet essentially acts as a traffic cop for the application flow: when the servlet receives a POST request, it records a survey response and updates the summary bean; when the servlet receives a GET request (from a private Web page requesting a summary status), it displays a statistical summary of

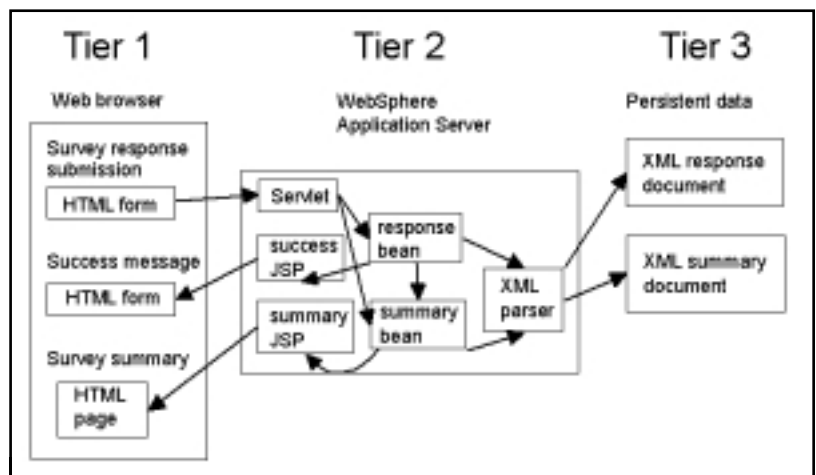


FIGURE 2 Three-tiered Web application model

all responses (see Figure 3). When the application starts, the summary is initialized from the summary file (on the back-end tier), if it exists. Beans for the responses and summary are used to store state data and write out XML to persistent files on the back end. To assist with the parsing of the XML data from the back-end tier, a SAX-based XML parser also runs on the application server. When the application is closed, the summary bean writes out its state data to the summary XML document.

By processing XML on the middle tier, the Conference Survey application is well placed to exchange data with other XML-based applications. If multiple applications reside on the same application server, events in one appli-

cation could trigger listener beans in another application to read its state data and subsequently mine it differently. For example, suppose another application wanted to take the opinions voiced by a survey respondent to help build a customer profile. The data could be used to target specific kinds of Web advertising to users based on the preferences they cited in their survey response.

On Tier 3, using XML documents as the vehicle for persistent data, you maintain the flexibility to mine the data as you deem fit without being forced down any particular path in terms of how it will be presented. For the amount of data being represented here, using a relational database would be overkill, while using serialized beans is overly complex. Because XML is self-describing by definition, and

XML parsers are readily available, using XML is preferable to a flat file (which might use, for example, name-value pairs), where the programmer is forced to write a custom parser.

The XML fragment in Listing 1 illustrates the data that's written out to a file for each response.

The technology used in the Conference Survey example provides a scalable model for building three-tier Web applications. If the amount of persistent data grows very large, XML files could be stored in a relational database. As additional kinds of data mining could be applied to the data, additional servlets and beans can be deployed to the application server. As popular Web browsers standardize on how to handle XML, then XML and XSL stylesheets could eventually replace HTML pages and JSP pages.

Although the application architecture is straightforward enough, the challenge remains to find a suite of tools that can work together to let you easily develop and test the end-to-end application. Enter VisualAge for Java.

Integrated Tools

Using VisualAge for Java, version 3.5, you can develop, test, and debug full-blown XML-based Web applications, all within the VisualAge for Java environment. Most development tool suites don't provide you with adequate integration. For example, you might have a great editor and compiler, but you need to deploy your code to a Web application server for testing and debugging. Unless these tools are integrated, you'll spend a lot of time redeploying code for testing rather than testing your code in a development environment. VisualAge for Java includes tools to create and run end-to-end Web applications like the Conference Survey application.

- **WebSphere test environment:** This runtime environment is a version of the IBM WebSphere Application Server, Advanced Edition product, streamlined for use specifically inside VisualAge for Java. The environment includes tools to start, stop, and configure the server (see Figure 4). If server-side source code is changed, VisualAge for Java incrementally compiles the code and hotlinks it into a running program.

You can run multiple, end-to-end Web application scenarios in this environment, such as e-commerce (store and customer applications) or B2B (supplier, vendor, and manufacturer applications). You can also pass data between the various applications, all from within the comfort of the WebSphere test environment.

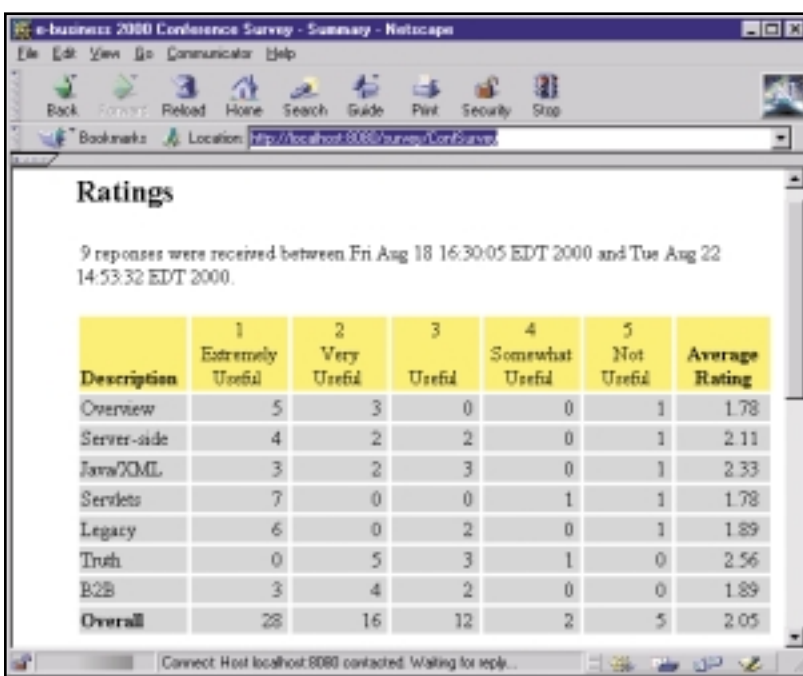


FIGURE 3 Conference Survey summary page

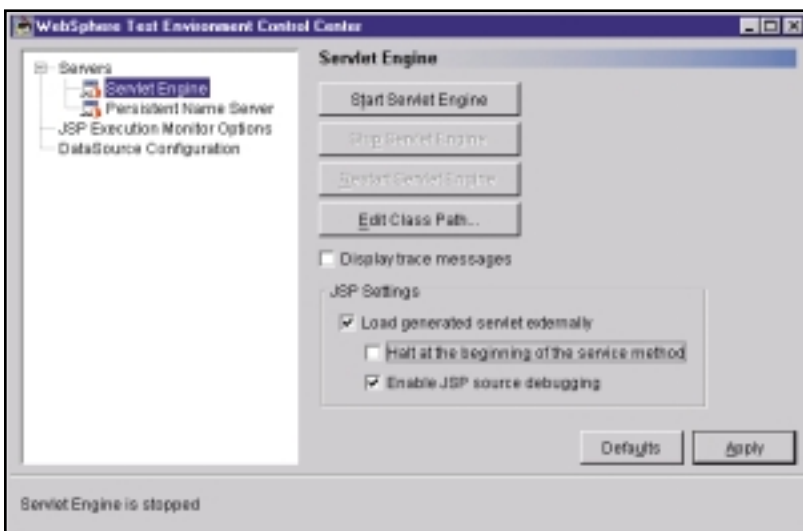


FIGURE 4 Control Center for WebSphere test environment



FIGURE 5 The Create Servlet SmartGuide

- **Servlet wizard:** The Create Servlet SmartGuide (see Figure 5) provides a quick way to create servlets and related Web resources files (HTML and JSP pages).
- **XML Parser for Java:** This parser is a version of IBM's XML4J parser, which conforms to standards and recommendations for XML, DOM, SAX, and Namespaces support.

In the Conference Survey application, the parser is used when the servlet starts. It reads the summary file, if it exists. If any responses aren't reflected in the summary file, the parser reads in those records and modifies the summary bean, as appropriate.

- **XML Generator:** This tool generates sample XML documents based on a DTD you provide. While an application like the Conference Survey is too simple to require this tool, you could use the XML generator to generate a diverse range of test cases.
- **Integrated debugger:** This debugger includes the features found in all good debuggers: breakpoint setting, variable inspection and modification, step into and through methods, multithread support, define expressions to watch as you step through programs, and so on. In addition, you can halt execution in the middle of a running program, change a variable value, and resume execution. In a complex scenario with multiple applications that pass data between each other, this debugging support can speed your development considerably. (see Figure 6).
- **JSP execution monitor:** This tool provides an integrated run-time view of your JSP page, letting you quickly debug problems arising from JSP-

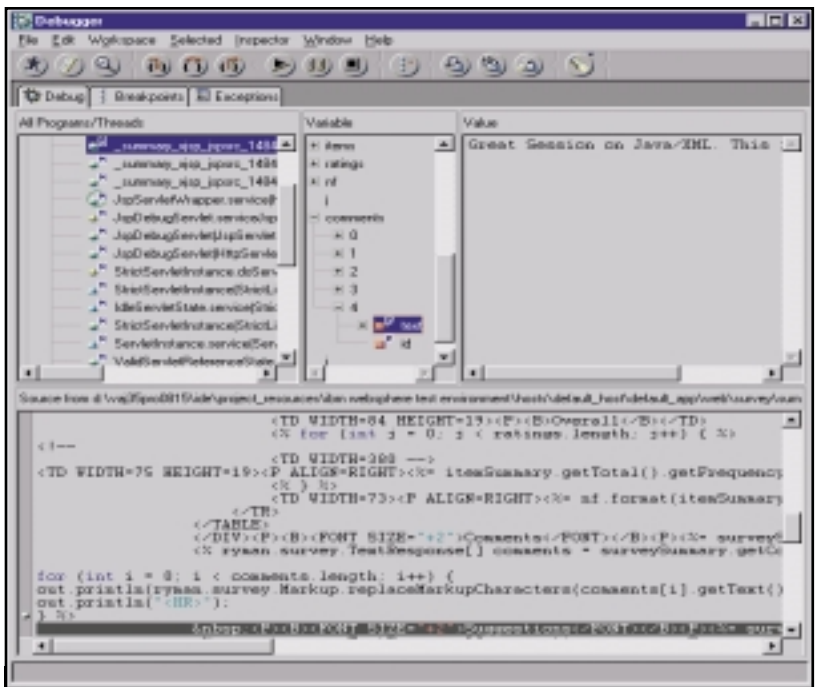


FIGURE 6 Integrated debug support

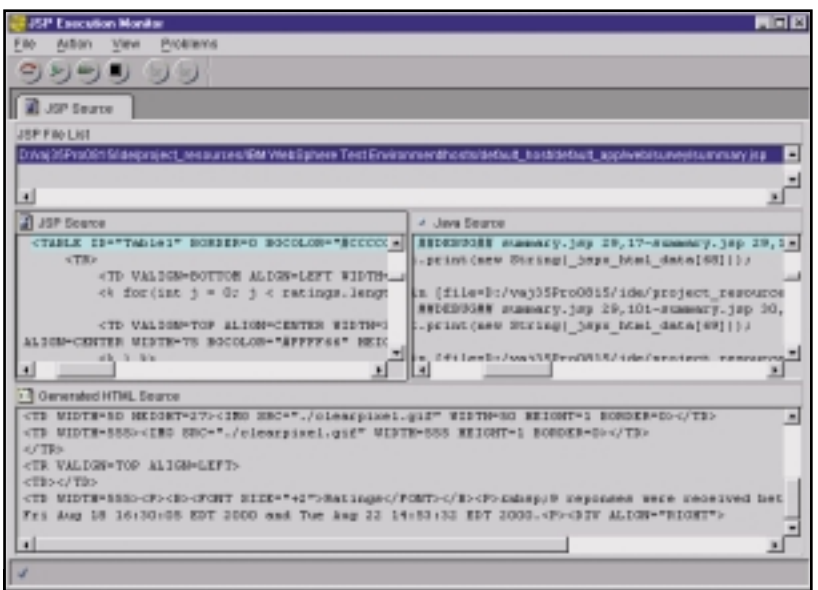


FIGURE 7 The JSP execution monitor

generated code. As with a debugger, you can step through a JSP page and see both the generated intermediary Java source and the generated HTML (see Figure 7).

Setting Up the Application

To set up the end-to-end environment in VisualAge for Java, we need to follow these steps:

1. **Load all necessary projects into the workspace.** Add the IBM WebSphere Test Environment, IBM XML Parser for Java, and Sun Servlet API. Of course, your application project will also need to be in the workspace.

In the response bean an instance of IBM XML Parser for Java is created:

```
String parserClass =
"com.ibm.xml.parsers.SAXParser";
parser =
ParserFactory.makeParser(parserClass);
parser.setDocumentHandler(handler);
```

Note that the SAX-based instance of this parser has been tested to work with the WebSphere Test Environment and is not guaranteed to work with other application servers.

2. **Edit the WebSphere Test Environment's servlet initialization file.** On server

start-up, you can specify that the WebSphere Test Environment initialize a specific servlet instance with particular parameters. In the default_app.webapp XML file, we've added a servlet entity for the SurveyServlet servlet. The entity specifies the directory where the application will be writing an XML file for each survey response (see Listing 2).

3. **Add all client-side Web resources (HTML, JSP, GIF) to the WebSphere Test Environment's default Web directory.** In the Conference Survey application we add the HTML and JSP pages, and all associated artwork files, to the <product_root>\ide\project_resources\IBM WebSphere Test Environment\hosts\ default_host\default_app\web directory.

AUTHOR BIOS

Luc Chamberland is a development manager in the IBM Toronto Lab, where his team develops XML parsing technology, including the XML4J and XML4C parsers. He previously worked on the VisualAge for Java team and has authored several papers on VisualAge for Java.

Arthur Ryman is a senior technical staff member at the IBM Toronto Lab where he is the B2B and XML tool architect. Previously he was the VisualAge for Java Solution architect, specializing in tools for servlet, JavaServer Pages, XML, and WebSphere development. Arthur is currently working on tools for SOAP, Web Services, and Internet Business Process Integration.

Running the Application

Now you're ready to run the application.

1. **Start the servlet engine in the Servlet Engine page of the WebSphere Test Environment Control Center.** The Console window provides status information on the servlet engine as it starts.
2. **Open a Web browser to the server-based URL (not file-based!) where the survey Web client resides** (see the URL in Figure 1). The WebSphere Test Environment uses port 8080 as the default port. In our example, the URL is `http://localhost:8080/survey/survey.html`.
3. **Fill in the survey and submit it.** When you receive the "success" message in the browser, you've essentially gone to the back end of the application and back: the servlet processed the submission and created an instance of a response bean; the bean wrote out response information to an XML

response document (in the directory specified in `default_app.webapp`); and a JSP page returned a status message back to the browser.

Without having to bother with deployment, we've been able to test a server-based application end to end using VisualAge for Java. But what if something goes wrong? Is there integrated debug support?

Debugging the Application

Problems inevitably arise during development – even to the best of us! Perhaps the code that creates the summary bean isn't properly storing the data it's parsing from an XML response document. In Figure 6, we've set a breakpoint where the summary bean is assigning values to a variable that holds parsed comment text. We step into the `setComments()` method and find the problem: instead of appending comments in the comment summary, the comment summary contents are replaced with current XML document contents. Modify the method and retest.

Consider a situation in which the summary page displayed in the Web browser isn't quite right. You need to debug your JSP file, but how? Notice that in the Control Center (see Figure 4) when we started the servlet engine we enabled the "JSP source debugger." This lets us debug the JSP source using the IDE debugger. You can inspect variables generated dynamically from the JSP source. On another page of the Control Center, the JSP Execution Monitor Options page, select the "Enable monitoring JSP execution" option, and the JSP Execution Monitor will appear the next time the JSP page is invoked. With the JSP Execution Monitor we can step

through JSP code to see the corresponding generated Java source and generated HTML (see Figure 7).

Extending the Application

Other information can be mined from the survey data:

- Calculate survey scores for each response based on some kind of weighting system. Store the score as part of the XML data for each response. Write a separate servlet that extracts the comment text from the high-scoring responses. The conference organizers could then look for inspiring quotations they could use next year.
- In the same manner, calculate session scores for the summary. The results provide information on which sessions and speakers were popular – and which should be avoided!

What if you want to add more entities to the XML response documents but don't want to go back through the data to pad the records? Just write out the new entity into new records and add some code to any bean handlers to assume a default value if the entity doesn't exist in a given XML document.

Conclusion

Using Java and XML, our sample application remains extensible. Using VisualAge for Java, you can quickly prototype Web applications. Servlets, JSP pages, an XML parser, and XML files can be used together in the VisualAge for Java development environment, making development and debugging a snap. 🍌

lchamber@ca.ibm.com

ryman@ca.ibm.com

Listing 1: Sample XML response document

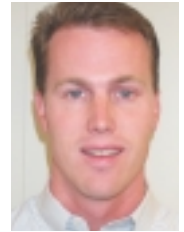
```
<?xml version='1.0' ?>
<SurveyResponse>
<ID>2</ID>
<Timestamp>Monday, August 21, 2000 5:07:32 PM EDT</Timestamp>
<ItemRatings>
<ItemRating>1</ItemRating>
<ItemRating>2</ItemRating>
<ItemRating>1</ItemRating>
<ItemRating>1</ItemRating>
<ItemRating>3</ItemRating>
<ItemRating>2</ItemRating>
<ItemRating>2</ItemRating>
</ItemRatings>
<Comments>Great session on Java/XML. This is the way of the
future.</Comments>
<Suggestions>more on Linux and Open Source stuff</Sugges-
tions>
</SurveyResponse>
```

Listing 2: WebSphere Test Environment start-up configuration file for servlets

```
<?xml version="1.0"?>
```

```
<webapp>
<name>default</name>
<description>default application</description>
<error-page>/ErrorReporter</error-page>
...
<servlet>
<name>SurveyServlet</name>
<description>Conference Survey servlet</description>
<code>ryman.survey.SurveyServlet</code>
<init-parameter>
<name>servletDirectory</name>
<value>${server_root}/hosts/default_host/default_app/web/sur-
vey/data</value>
</init-parameter>
<servlet-path>/survey/ConfSurvey</servlet-path>
<autostart>true</autostart>
</servlet>
...
</webapp>
```





Interview...with Ted Farrell

CTO OF WEBGAIN

A DISCUSSION OF THE STATE OF JAVA DEVELOPMENT TOOLS



Java 2 Enterprise Edition has made Java a full enterprise-scale language – in all senses. Improvements in functionality, scalability, and performance have made it possible for more and more enterprise applications to be undertaken with Java technology.

Java projects are growing both in terms of complexity and the amount of code required to complete them. Developers need to turn to the types of methodologies and tools they've used with other development technologies – including modeling.

Modeling is a visualization process for constructing and documenting the design and structure of an application. As a matter of good practice, developers should make at least some outline of their application to show interdependencies and relationships between application components and subsystems. Modeling tools make this possible. Because modeling is dynamic, as one change is made in the model, the developer can see the ripple effect of the changes throughout the project. Modeling can also be introduced in the middle of an existing project; most modeling tools can read existing code and create a visual model based on the existing code base, giving developers a high-level view of what would otherwise be thousands of individual lines of code to review.

The Universal Modeling Language (UML) is the standard language used by the many modeling tools on the market. UML was designed to unify the many proprietary and incompatible modeling languages and create one modeling specification.

Modeling tools are increasingly being applied to Java development projects. Still, more than a few programmers use a text editor, and many developers haven't even thought about using modeling to create their applications. With the increasing complexity of enterprise Java applications and Java components, modeling will become a necessity for reducing development time and ensuring that a program is well written the first time around.

In a recent question-and-answer session, Ted Farrell, chief technology officer of WebGain, talked about his view of the growing role of modeling in the development of Java applications.

Q: At what point should you, as a developer, say you really need UML?

A: UML diagrams are designed to capture abstract thought before the programming begins. A use case or sequence diagram is a common starting point for many of our developers. You can model a flow of a method or several methods, and when programmers look at that diagram, they can see how it would translate into functional calls in code.

The larger your application and the more people involved, the more apparent the need is for a UML modeling tool. But that decision isn't a definite line. The size

of the application you're building and the number of people involved could be used as a driver of whether or not a UML modeling tool is a good choice for your project.

Q: How portable are models? Can I take something from Rational Rose and read it into any UML tool and vice versa?

A: It's getting there. There's a standard developed. It's an XML standard called XML, or XML Metadata Interchange Format. It's UML described in XML. The next release of WebGain's UML tool, WebGain StructureBuilder, will support reading and

writing XML files. Rose right now has utilities to do that. This will be a good interface for exchanging XML information between different products.

Q: What are some of the features to look for in a modeling tool?

A: For the Java developer, it comes down to how in touch the modeling tool is with the Java language and the code they are developing. The further away the tool is from the actual language and development environment, the harder it is for the developer to use. Most UML tools are language agnostic. WebGain StructureBuilder is concentrated only on Java. You're talking in Java terms, not pure object-oriented terms anymore. You're talking in terms that any Java programmer would know as part of the language. They don't have to do that mental translation in their heads. If the tool is talking in generic object terms or doesn't keep in sync with the code, it becomes a lot more difficult for the programmer.

Programmers need a UML tool that is aware that Java code is present during different phases of the development cycle, and that the modeling isn't only done at the beginning of a project. Today, a lot of design is done up front, but as the developer starts coding, the application evolves. With Internet demands causing development cycles to shrink from years to weeks, a development environment must allow for rapid design, prototyping, and coding, while maintaining the integrity and quality of the code being developed. If you have a UML tool involved throughout the development cycle, you will have fewer deviations between your application and initial design. You will be able to identify differences during the coding phase, and either adjust your code or your design to meet the end requirements.

Q: How does modeling help with commercial or prebuilt components?

A: It doesn't make a difference whether it is software you are writing from scratch, or applications you are assembling from existing components and/or services, or both. Good design is always needed. Understanding the requirements of your

software, and being able to articulate the design and flow of the application you are building, is essential. UML modeling tools allow you to do this in a consistent and effective manner.

Q: What role does modeling play in reuse of code?

A: The better the design of any code, whether it's a component or a standard software, the better and the more reusable the code will be. On the flip side, when reusing code, modeling can be invaluable in helping you understand the boundaries and dependencies of the different components you are using in your application.

Q: Can modeling be used to reverse engineer Java source code that wasn't written by a modeling tool and no model exists?

A: Absolutely. This is a very valuable feature. Developers and consultants use WebGain StructureBuilder to do this all the time. If they're going into an environment where they're not familiar with the code, or sharing code across a development or distributed development environment, a feature like this can literally save days worth of time. They can read in large amounts of code and WebGain StructureBuilder will build a set of diagrams for them. This helps them quickly visualize the areas of the code and what they do and lets them see how objects relate to each other.

Q: How much design information can you get from undocumented code?

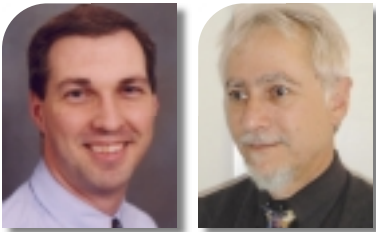
A: Most of the structural information of the design can be obtained from the Java source code. Most of the relationship information is there. Information like whether a relationship is aggregate or composition can't be obtained from the code and would need to be identified by the user in the UML tool, but other than those, and other similar items, you can get a pretty decent representation of your object model straight from the Java code.

—continued on page 120

Automated Software Inspection

Save development costs and improve reliability of Java applications

WRITTEN BY
SCOTT TRAPPE AND
LAWRENCE MARKOSIAN



Unfortunately, most conventional test methodologies assess only about 60% of the code in any Java application. And as applications become larger and more complex, even less code is covered by conventional test methodologies. So the challenge facing Java developers remains: How do you effectively debug applications before you deploy them?

New automated inspection techniques are becoming available from independent QA providers that make it practical to inspect and evaluate source code before you test. In fact, inspection at various stages of development can isolate critical, crash-causing faults before Java applications are compiled and tested, resulting in higher quality code with less reliance on software testing. Unlike testing, automated software inspection (ASI) can cover 100% of Java source code in a fraction of the time.

ASI is a new approach that provides early feedback to developers about crash-causing, data-corrupting software defects. ASI reads the source code, analyzes the structure of the software, determines how data flows through the program, and identifies code-level defects such as array bounds violations, dereferencing null objects, and race conditions. For example, conventional procedures don't test every possible application thread interaction in a multithreaded Java application. ASI, on the other hand, can provide complete code coverage, examining every thread for points of potential race conditions. Similarly, testing for out-of-bounds array

access errors is impractical since it requires developing test cases that cover every possible access. Here, ASI can perform a comprehensive static analysis without having to test every array access and can uncover potential problems before testing. ASI requires neither target hardware nor test cases, and thus can detect errors that would escape conventional testing. It's ideal for isolating hard-to-find problems in Java applications.

First Inspect, Then Test

It's always best to find defects before they become bugs – ideally, immediately after coding. One of the best lines of defense in improving the overall quality of software is to improve software testing. However, while testing is acknowledged to be a critical part of total quality assurance, it has limitations:

- It's expensive and time-consuming to create, run, validate, and maintain test cases and processes.
- The number of statements tested, that is, code coverage, inexorably drops as the system grows larger. The bigger the application, the less source code that testing can validate.
- Even when testing uncovers a fault, it's difficult to trace a failure from a test case back to the code causing the problem.
- Testing can't uncover all potential bugs. A study conducted by Capers Jones concluded that even the best testing processes would remove, at

most, 85% of all software defects. Additional research conducted by the Standish Group determined that most QA organizations are only 30–40% effective at identifying software defects.

The objective in conventional testing is to perform complete code coverage by guaranteeing that every statement in the system has been executed at least once by a given test. Figure 1 illustrates the problem with a diagram of a single function that has a single entry and, for simplicity, a single exit. Execution begins at the top and exits at the bottom. Each box represents a basic block, also with a single entry and exit. The solid lines connecting the boxes represent possible flows of control.

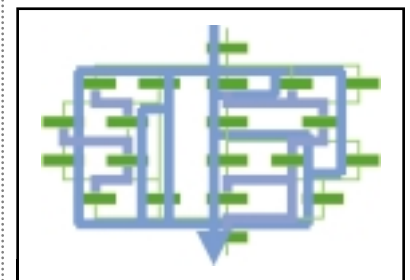


FIGURE 1 Function with 100% code coverage

The lines show the execution paths of several test cases. In the ideal case shown here, every statement was executed, achieving total code coverage, although such comprehensive testing is

rare. And even if total code coverage is achieved, many defects can remain. Effective testing requires total path coverage in addition to statement coverage, so all potential paths that connect two statements should be tested. The lines in Figure 2 show several paths that were missed by the test cases.

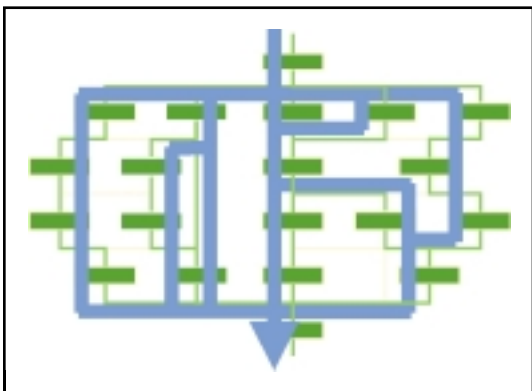


FIGURE 2 Code coverage without effective path coverage

ASI can minimize this problem by providing both code coverage and path coverage. Experts agree that inspecting software before testing is the most effective way to find software faults. Studies by Capers Jones show that up to 65% of all coding defects can be identified prior to testing, thus realizing substantial savings in both time and dollars.

The Cost of Debugging

While ASI isn't a replacement for testing, it is a cost-effective, complementary QA technique that shortens time-to-market and increases reliability for Java applications. Applying automated inspection early and at critical stages in the development process saves development time and resources.

Outsourcing inspection to a qualified service provider may seem expensive. Some developers have considered trying to do their own software inspection, but this is truly a specialized skill requiring a high degree of technical sophistication, not to mention substantial dedicated computing power. Still, the savings you can realize by using a software inspection service can be substantial. Consider the following examples.

Let's review a typical Java application consisting of 500,000 lines of code. Most applications average one software defect for every 1,500 lines of source code, so you'd expect our Java application to have about 330 defects. The cost of thoroughly inspecting a Java application of that size using ASI would be about \$50,000.

That may seem like a substantial investment, but consider the costs of alternative approaches, such as inspecting the same application manually. It would take a good programmer an average of about five hours of dedicated work to isolate a single software problem. The typical software developer costs the company approximately \$8,500 per month, and there are about 170 work-hours per month. That averages out to a cost of \$250 per defect, or a total cost of \$82,500 to manually inspect a 500,000-line application, and it would take about 10 man-months. What that estimate doesn't take into account is the loss of productivity and additional loss of revenue resulting from dedicating valuable development time to inspection. ASI, on the other hand, costs about \$50,000 and would take less than two weeks. And ASI provides higher accuracy with repeatable results.

Now let's consider the same example and the cost of automated inspection versus testing. A typical QA engineer costs the company about \$6,000 per month, which averages out to a cost of \$1,600 to find each of our 330 defects. The average QA engineer can isolate only about eight defects per month, so thoroughly testing 500,000 lines of code would take about three man-years at a total cost of about \$250,000. ASI could be done in two weeks at 20% of that cost.

Finally, let's consider the cost of ASI versus improving the defect removal rate. It costs an average of \$14,000 to isolate a bug after the software is deployed (according to the Cutter Consortium), and there is an average of 5,000 bugs introduced before inspection or testing. Even assuming you have the best possible average defect removal rate of 85%, that still leaves an average of 15% or 750 bugs that will make their way into the first commercial release. Even if ASI improves the defect removal rate by only 5%, that still represents a potential savings of \$3.5 million.

Clearly, ASI can represent substantial savings in Java development. But how would you apply ASI to isolate bugs unique to Java? Here are a few examples.

Types of Java Defects Found by Automated Inspection

Since Java is considered to be a relatively "safe" programming language that performs a lot of runtime checking, many programmers believe that extensive testing is sufficient to guarantee high-quality code. Not so. When Java performs a runtime check that detects an exception condition, it throws an exception. If the programmer hasn't

handled the exception properly, the program will either crash or the environment will be corrupted in a way that prevents normal continuation. Even if the exception is handled, it's likely that the computation performed is incomplete and may cause data corruption or further exceptions. Even with extensive testing, you can still get latent defects that will crash an application in the field. Finally, threading problems are a serious issue in many Java applications and are extremely difficult to isolate with testing. While mechanisms in Java, such as synchronization of shared resources, can prevent some of these problems, the defects themselves remain. ASI can uncover these faults.

In addition to finding the point in the code where an exception or other problem can occur, such as a null object dereference, ASI can provide the information needed to correct the underlying problem (for example, the place in the code where a null object can be generated). Testing places a greater burden on the programmer to interpret the raw results and trace back to find where the null pointer was generated.

Java defect classes that we expect to detect are a subset of Java runtime exceptions plus threading defects. We haven't made a final determination of which defect classes will be implemented first. Likely candidates include:

- *NullPointerException*
- *IndexOutOfBoundsException*
- *NegativeArraySizeException*
- *SecurityException*
- *Race conditions*
- *Deadlock*

No tools currently do a good job of detecting these defect classes without extensive testing (and who really does "extensive" testing?). Several inspection tools are available, but they find relatively uninteresting defects (such as coding standard violations) or they don't report real defects accurately.

The trick in defect detection is to isolate only the defects that are truly important: crash-causing faults and other problems that warrant allocating the resources to fix them. This rules out problems such as:

- *Violations of most coding standards*
- *All false positives*
- *All "low value" defects*

The false positives are simply code fragments that look like defects but really aren't. They're usually caused by the inability of testing tools to perform a sufficiently deep analysis of the program to rule out the apparent defect. An example would be a failure to check for a zero

divisor prior to a division by zero. If you just examine the code in the particular method where the division is performed and you find no check for a zero divisor, you might be tempted to report a defect in the method. However, it may be that the check is performed elsewhere so the method is never called with a zero value for the divisor. In this case reporting a potential arithmetic exception for division by zero would be a false positive.

Another useful characterization of defects is "low value." These are true defects but, for whatever reason, they're of little or no interest to the developer. For example, in C and C++, a null pointer dereference on an out-of-memory condition is a true defect, but it's often (although not always) a low-value defect because the expectation is that the application (and maybe the system) is about to crash anyway.

Java coding can cause a number of problems where inspection will isolate crash-causing faults that testing can miss, creating a false negative. We'll also consider a similar example that seems to be a problem, but on closer inspection isn't. Our examples are about race conditions in Java.

Listing 1 is a synchronized method that adds an element to a table.

Each thread calls this method at a different time. The method locks the table, creates a new table entry, recalculates the column sizes by calling `setMaxColSize`, and then adds the new row to the table. The code for `setMaxColSize` is shown below and sets a number of instance variables.

```
private void setMaxColSize(Table t) {
    DateTime =
    t.dateTime.toString().length();
    Id = t.Id.length();
    Status = t.status.length();
}
```

This application has an event-driven user interface. At any point in the execution of the above code, a user could press a button in a frame causing another process to execute. In the File-Save operation, for example, which is invoked by pressing a button, there is a call to a method called `getMaxColSize`. This method isn't synchronized.

Thus, it's possible that, while a thread is in `setMaxColSize` setting instance fields, the main event-handling thread could trigger a file save that would read those column sizes. With a bad interleaving the reading thread could get some new values and some old values for the column sizes:

```
public int[] getMaxColSize() {
    int[] verytemp=
    {DateTime, Id, Status};
    return verytemp;
}
```

This is a potentially dangerous situation that can cause unexpected results. Where it's hard to detect this race condition using conventional programming testing, inspection would identify this kind of problem.

A False Positive

It's common practice in thread programming for the programmer to rely on each thread retaining a copy of all required variables rather than using synchronization to share memory between threads. This assumes that sharing variables isn't required within the application. This is an acceptable way of avoiding race conditions, assuming that none of the variables accessed by the threads are static or in any way accessible by multiple threads.

The example in Listing 2 is adapted from Eckel's *Thinking in Java*. It relies on three variables: `countdown`, `threadcnt`, and `threadnum`. A perfunctory analysis of this code might conclude that a potential race condition exists. However, each thread has its own copy of two of these variables. The `threadcnt` variable looks a little more troublesome because it's static, but note that it's modified only in the constructor, not in a start or run method, so multiple threads won't access it. Again, here's a nonfault that might be incorrectly identified as a fault during a superficial inspection.

Conclusion

From these examples it's clear that automated software inspection can be extremely valuable in Java development. Testing can uncover many problems, but it's often difficult to assess the severity of the defects and trace them back to their source. Automated software inspection, on the other hand, provides more complete coverage of Java code, not only identifying many defects earlier in the software development life cycle but also providing information necessary to correct the problem.

ASI isn't a replacement for testing. However, it's a cost-effective, complementary QA technique that shortens time-to-market and increases software reliability. Applying automated inspection early in the development process – as soon as a build can be done – and at various strategic later stages, saves time and resources. It also saves developer time and allows for creation of higher quality applications. ☘

scott.trappe@reasoning.com

lawrence.markosian@reasoning.com

AUTHOR BIOS

Scott Trappe is president and COO at Reasoning, Inc. (www.reasoning.com), an application service provider specializing in software quality and modernization.

Lawrence Markosian, a founder of Reasoning, is product manager for its automated software inspection service.

Reasoning is headquartered in Mountain View, California, with sales and service locations in North America, Europe, and Japan.

Listing 1

```
/**
 * Synchronized method to add an element.
 */
public void addElement( Client incomingClient )
{
    synchronized( lock ) {
        Log query = incomingClient.getLog();
        results = query.getAllSearchResults();
        for ( int k = 0; k < results.length; k++ ) {
            ...
            setMaxColSize(XTable);
            tableData.add(XTable);
        }
    }
}
```

Listing 2

```
public class Simple extends Thread {
    private int countdown = 5;
```

```
private int threadnum;
private static int threadcnt = 0;

public Simple() {
    threadnum = ++threadcnt;
    System.out.println("Making "+threadnum);
}

public void run() {
    while (true) {
        System.out.println("Thread " +
            threadnum+ "("+"countdown+"");
        if (--countdown == 0) return;
    }
}

public static void main(String args[]) {
    for(int i=0; i<5; i++)
        new Simple().start();
}
```



GET YOUR OWN!

Special TechWave '99 Issue!
 PowerBuilder Journal
 DEVELOPER'S JOURNAL
 WEB DEVELOPMENT DOES THE TANGO
 TANGO
 BUILDING ENTERPRISE PORTALS
 GOLD FUSION Developer's Journal
 XML: IT'S THE 'X' THAT MATTERS
 XML JOURNAL
 PRESENTING XML
 OUR SPECIAL INAUGURAL ISSUE!
 wireless MAGAZINE
 WHAT'S GOING TO BE THE NEXT THE NEXT CROP?

Call and Subscribe today!
 Get Your Own Subscription to the Finest
 Technical Journals in the Industry!
 1-800-513-7111 www.sys-con.com

Don't Miss JDJ's **LINUX** Focus Issue!

Coming in December!



**For Advertising
 Information Contact:**
 Carmen Gonzalez
 Vice President, Advertising Sales
 Java Developer's Journal

**201 802-3021
 or email
carmen@sys-con.com**

www.JavaDeveloper'sJournal.com

Announcing...



Look for it on your newsstand
 and worldwide in
December

**DON'T
 MISS IT!**

SYS-CON Media,

the world's leading publisher of

Internet technology magazines for developers,
 software architects and e-commerce professionals,
 becomes the first to serve the rapidly growing
 wireless application development community!



www.wireless-magazine.com

iPlanet Application Server 6.0

by iPlanet E-Commerce Solutions

REVIEWED BY JIM MILBERY



AUTHOR BIO

Jim Milbery is a software consultant with Kuromaku Partners LLC (www.kuromaku.com), based in Easton, Pennsylvania. He has over 16 years of experience in application development and relational databases.

jmilbery@kuromaku.com



Test Environment

Gateway GPi, 196MB RAM, 30GB disk drive, Windows 2000

iPlanet E-Commerce Solutions

4220 Network Circle
Santa Clara, CA 95054
Phone: 888-786-8111
Web: www.iplanet.com

Sun's iPlanet division has become the umbrella organization for all the application development software that Sun has collected over the past several years. Sun's initial foray into this business was their acquisition of application-server vendor NetDynamics – but the core technology for iPlanet came from the Sun/Netscape/AOL multiplayer trade that put control of Netscape's server products under Sun's domain. Ultimately, Sun would go on to acquire Forté Software and NetBeans to round out their application server family of products. Technically speaking, Sun doesn't own the Netscape product line outright and iPlanet E-Commerce Solutions is an alliance among AOL, Netscape, and Sun that was formed in March 1999. The iPlanet name comes from another Sun acquisition and the alliance appears to be primarily controlled by the Sun camp. The iPlanet 6.0 application server is the latest release of the consolidated product suite – one that includes code from both NetDynamics and Netscape.

iPlanet 6.0 – Ready for Java

iPlanet considers the 6.0 product to be a “fourth-generation” release that builds upon a proven architecture. Since there are three principal bloodlines for iPlanet (NetDynamics, Netscape/Kiva, and Forté), the core technology has been around in some fashion for a number of years. However, the task of integrating these disparate products into a scalable and reliable platform is no small task.

Classically speaking, there are two separate paths that customers can take when implementing enterprise-class applications. They can buy all their software from a single vendor and take an integrated approach, or they can buy best-of-class products from lots of different vendors and assume the integration tasks on their own. While the Java 2 platform makes the best-of-class approach a more viable one, there are still hurdles to be overcome from a business standpoint when using the best-of-class approach. Take support, for example. The task of coordinating problem resolution among dozens of vendors can be a Herculean task. With iPlanet you get the integrated approach. Netscape has lots of software assets in the application server space in the form of directory services, Web servers, and application server technology. In a sense, the iPlanet product is itself something of a composite of a number of best-of-class products. The difference is that iPlanet has taken on the task of doing the integration work for you.

The iPlanet 6.0 product is the first product to pass the J2EE certification suite. Given iPlanet's cozy relationship with Sun, this comes as no major surprise. Taken from the J2EE perspective, iPlanet 6.0 offers comprehensive support for the J2EE stack as shown in the following features:

- Java 2 SDK
- EJB 1.1
- JDBC – J2SE
- JNDI 1.2
- Servlet 2.2
- JSP 1.1
- JavaMail, JAF 1.1
- RMI/IIOP
- Java Transaction API

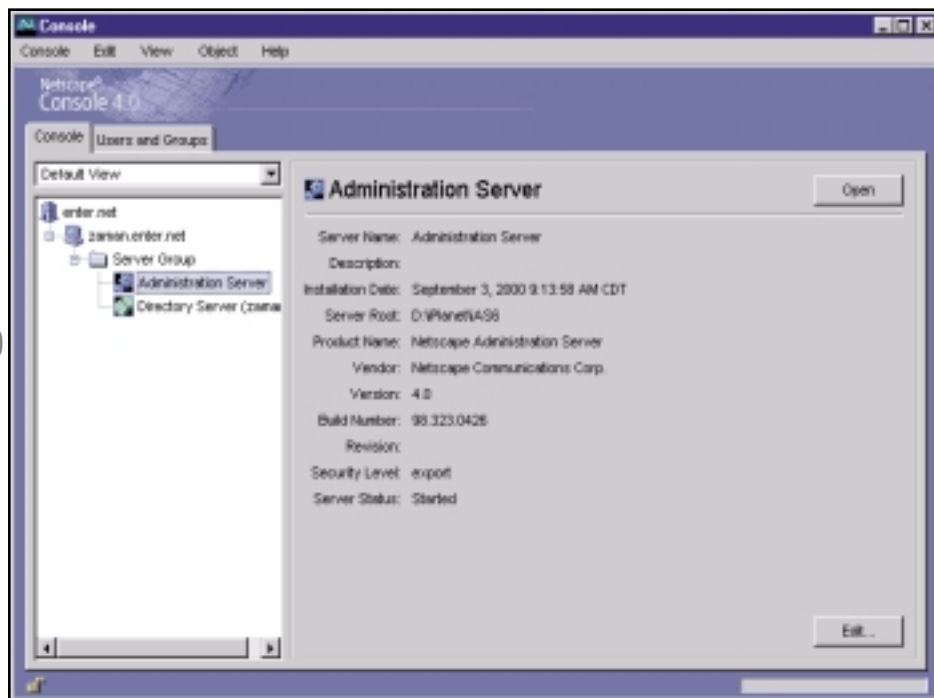


FIGURE 1 Netscape Console



Here's a sneak peek...

Java Technology on the Linux Platform

A Guide to Getting Started
by Calvin Austin

Java and Linux

Ten Feet Tall and Bullet-Proof
by Ben Okopnik

Linux

The Perfect Java Development
Environment
by Micah Silverman

Guest Editorial

Heres one, take a penguin and a duke...
by Alan Williamson

Java and MySQL

mySQL/Java, a Harmonious Relationship
by Mark Matthews

Apache and JServ

RelationJava Servlets courtesy of
JServ/Apacheship
by Alan Williamson

Embedded Linux

Do it yourself embedded Linux
by Marcel Gagné

Don't miss the December issue!

Next
NOVEMBER

Affinity

2/3

p/u

from CFDJ

One of the key underlying components of the iPlanet application server is the Kiva Engine (which formed the core of the Netscape Application Server product as well). Kiva initially provided strong support for C/C++, and many applications that were built with the Kiva/NAS rendition of the product relied upon this support. With the release of iPlanet 6.0 the migration to Java is in full swing. While you can still use C/C++ with iPlanet 6.0, many of the C/C++ features have been deprecated – the handwriting is on the wall as far as the C++ language goes. Clearly, iPlanet 6.0 has been designed to favor the Java developer.

Installing and Working with iPlanet 6.0

I downloaded the Windows NT version of the application server and application builder from the iPlanet Web site. The iAB evaluation kit includes a built-in copy of the server, so it's not necessary to download them separately to work with the product on Windows NT. iPlanet 6.0 offers tight integration with the iPlanet Directory Server, which became evident during the server installation process. Given my lack of experience configuring directory servers, I accepted the defaults in most cases. As far as HTTP servers go, iPlanet supports the iPlanet Web server, Microsoft IIS, and Apache – but the installation script only provides automatic installation for iPlanet and IIS. After installing the application server, I went through a second installation to get the Netscape Console installed (see Figure 1).

The iPlanet Console, which is written in Java, is the interface that allows you to manage application servers and directory servers. Personally, I prefer to have a rich-client interface like the iPlanet Console with which to manage application servers. I've found that there are too many moving parts for a plain vanilla HTML interface to handle. The iPlanet console is Java-based and is relatively easy to work with, and the interface itself was surprisingly crisp. Once I had the server running, I had some small problems getting iAB itself installed, including some minor problems with the tutorials.

The iPlanet Application Builder comes straight from the original Netscape Application Builder product (which in turn came from the original Kiva product). iPlanet positions iAB as a tool for entry-level development and recommends their own Forté for Java Enterprise Edition IDE for more complex applications. iPlanet

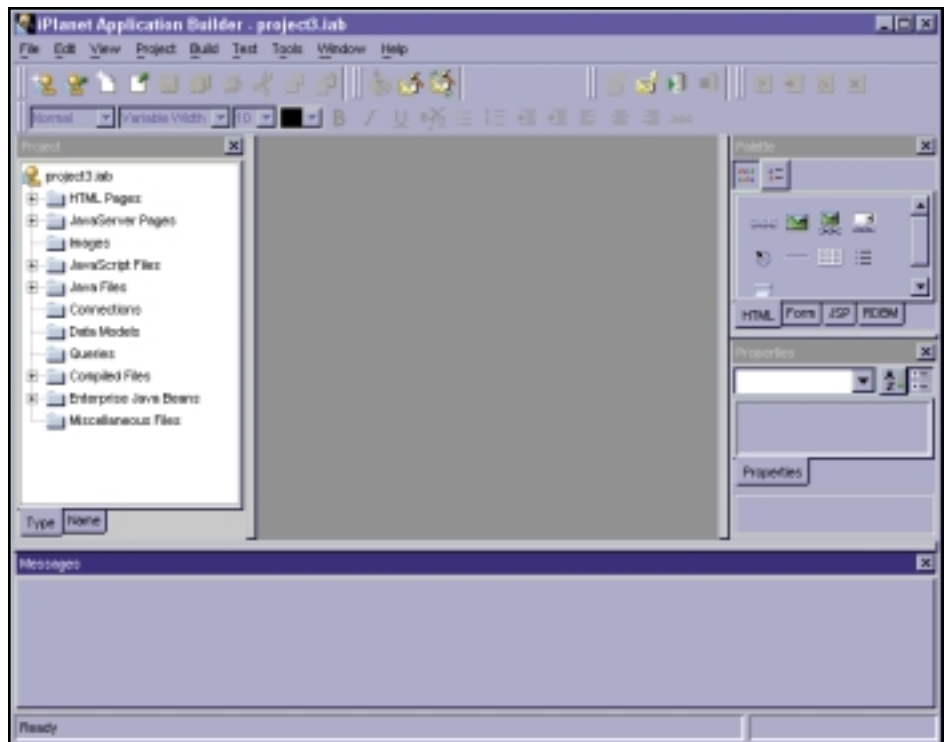


FIGURE 2 iAB has many of the features of the most popular IDEs.

also has interfaces for a number of third-party Java IDEs such as JBuilder, VisualAge for Java, and WebGain. iAB in its current form has support for all the critical features of iAS. It sports a number of wizards for managing complex tasks such as building database forms, displaying result sets, creating logon forms, and generating session/entity beans. iAB has many of the features of the most popular IDEs including dockable windows, project management tools, and a number of wizards (see Figure 2).

In many ways iAB is a lot like IBM's WebSphere Studio. Both interfaces have been designed to make it easier for a Java developer to create JSP and Servlet applications with connections to RDBMS engines. iAB wasn't meant to replace an enterprise-class IDE; this is evident if you're an experienced Java developer. (For example, team development and sophisticated debugging aren't provided within the iAB environment.) In addition, there are also some legacy quirks in iAB (such as the need to create Kiva data-model files) that might seem counterintuitive for developers that have experience working with a more complete Java IDE. iPlanet doesn't position iAB as a replacement for an industrial strength Java IDE, but rather as an entry-level tool for building applications. However, if you're not familiar with all the ins and outs of developing a J2EE-style application, iAB is a good starting point. Rather than relying on the examples that come with the installation kit, I'd recommend that you download the latest samples from the iPlanet Web site. In fact, the best demonstration application is the Pet Store application that's

described in the J2EE Blueprint document. (You can download the example application along with the J2EE, and instructions for deploying Pet Store to iAS can be found on the iPlanet Web site.) From the server standpoint, iPlanet has packed a ton of features into the server itself, including XML parsers and an XSLT engine.

iPlanet provides a complete e-commerce platform, and iPlanet Application Server forms the transactional basis of that platform. iPlanet also provides a number of other products that connect with iAS. In addition to the iPlanet Web Server and Directory services products, iPlanet also offers an iPlanet Process Manager, Messaging Services, and a Wireless and iPlanet Portal Server to work with iAS. The resulting group of products is an impressive suite from a features perspective. At least on paper, iPlanet has one of the most complete application server suites on the market.

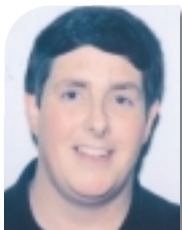
Summary

The transformation of NetDynamics, Netscape Directory Server, NAS/Kiva, and Forté into a single, unified product is well on its way with iPlanet 6.0. While there are still vestiges of the individual products scattered through iPlanet 6.0 (such as references to "kjs" and "kcs" for you Kiva developers), the product has come a long way in a relatively short time. iPlanet has packed a ton of services into a comprehensive J2EE server product. I'd recommend putting iPlanet 6.0 on your short list if you're committed to building Java-only applications. ☉

mBedded Server 4.0

by ProSyst

REVIEWED BY DON WALKER



AUTHOR BIO

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

don@enticesoftware.com



Test Environment

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

ProSyst USA
555 NorthPoint Center East
Studio 450
Alpharetta, GA 30022
Web: www.prosyst.com
Phone: 678 366-4607
E-mail: usa@prosyst.com

How many appliances do you have in your household? Unless you're currently a contestant on "Survivor" island, the answer is probably over 20. Refrigerators, microwaves, TVs, VCRs, dishwashers, vacuum cleaners, and the favorite appliance of so many programmers – the coffemaker. The list goes on. Unfortunately, although appliances have grown increasingly intelligent in recent years, they tend to exist in their own individual universes, oblivious to the world around them.

Wouldn't it be nice if our appliances could pool their resources and take advantage of the Internet? An automobile could use a cell phone to schedule maintenance and record the appointment in a personal digital assistant. A blood pressure monitor could immediately alert a hospital in the event of an emergency.

Although homeowners could take an active role, for example, by using remote access to turn an appliance on or off or to check the home security system, installation would have to be "Plug-and-Play." It would be too much to expect even the most technically savvy homeowner to link up appliances throughout the home, particularly since many appliances could interact in ways that were not even conceived of by the product manufacturers.

Despite this complexity, this is exactly the type of convenience we should expect in the near future according to ProSyst Software, the developers of mBedded Server 4.0 – a Jini- and WAP-enabled server designed specifically with embedded products in mind.

Device of Embedded Services

The job of orchestrating cooperation between a variety of appliances is made more difficult by the fact that the devices are likely to have been manufactured by different companies at different times and in different places. Adding further to the complexity is the fact that these devices may need to access external networks across the Internet (to schedule maintenance or report medical information, for example).

To help deal with this complexity, communication can be funneled through a single device in the home or office that's responsible for managing client requests on the local network, allocating resources, and providing a gateway to the Internet. ProSyst refers to this as the *device of embedded services*. mBedded Server 4.0 is a server infrastructure that can reside on or be downloaded to the device of embedded services in order to provide this functionality. Because mBedded Server 4.0 is 100% Java, it's platform independent. Consequently the device of embedded services can be any machine that offers a Java Virtual Machine (JVM).

Open Services Gateway Initiative

Non-PC devices require new means of exchanging data on the Internet. Not only does your blood pressure monitor need to communicate across the Internet with the hospital, but also, conversely, the service provider may want to install updated software on your local network.

The Open Services Gateway Initiative (OSGi) is a group of over 60 companies (including Cisco Systems, Compaq, Ericsson, GTE, IBM, Maytag, Motorola, Nokia, Oracle, ProSyst, Sony, Sun, Whirlpool, and others) that offer a solution. This solution revolves around the concept of a services gateway. The Open Services Gateway (OSG) is an embedded server that resides between the local and remote networks. The OSGi specification describes an API that can be used by network operators and device manufacturers for a variety of purposes, including managing the secure delivery of services from trusted service providers.

In this model, services provided by the various devices are packaged into components called *bundles* that can be installed, started, stopped, removed, and updated as necessary by the framework. Bundles can register services for other devices to use and look up ones for their own use.

The bundles are JAR files that contain resources needed by the service (such as Java classes) and a Manifest file used by the framework to activate the bundle.

Service Container Architecture

mBedded Server 4.0 complies with the OSGi specifications through its "Service Container" architecture. Devices communicate by sharing services. These services are provided by the devices themselves or installed on the server using the administrator applications provided with mBedded Server.

The services provided with mBedded Server 4.0 include:

- **Core services:** Login, log, and remote file
- **Administration:** Administration and user manager
- **Jini services:** Jini container and lookup
- **Mail services:** FTP client, IMAP, MIME, NNTP, POP3, and SMTP
- **Web services:** HTTP(S) (including Servlet support) and DNS client
- **Message services:** Message queue and publish/subscribe
- **Request services:** PMP (ProSyst Messaging Protocol), PMPTCP, request, and stack
- **Other services:** Crypt
- **Example services:** Printer and schedule

The power and flexibility of the system allow for the seamless addition of new services and the updating of existing services from a remote administrator application without shutting down the server. The ability to do remote administration is important since it's possible that the device of embedded services lacks a display.

This flexibility also makes it possible to configure the server to optimize disk space and memory usage – an important consideration in the world of embedded devices. mBedded Server 4.0 needs 100KB–500KB of disk space depending on the services required.

Creating a Simple Bundle

To demonstrate how easy it is to create, install, and use the services of a bundle, we can design a service that simply displays a string of text that's passed as a parameter.

The first step is to create an interface for the service.

```
public interface SimpleTextDisplayer
{

public void displaySimpleText(String
theText);
}
```

The second step is to create the class that implements the service and any support classes that might be needed. In this simple example only the SimpleDisplayImpl class in Listing 1 is required. This class merely provides a frame for a label that displays the text passed to displaySimpleText().

The third step is to create an activator class. The framework uses this class when the bundle is started, and is responsible for registering the service (see Listing 2).

The BundleContext object that's passed as a parameter to start() provides access to the framework and allows the bundle to call registerService() to register SimpleTextDisplayer as a service.

The fourth step is to create a manifest file. This file provides information about the bundle, including the name of the activator class that assists the framework in starting the bundle and the names of other packages this bundle relies on to provide its service or services.

```
Bundle-Vendor: JDJ
Bundle-Version: 1.0
Bundle-Activator: SimpleDisplayActi-
vator
Bundle-Name: SimpleDisplay
Export-Service: SimpleTextDisplayer
```

Finally, the three class files and the manifest file are packed up in a JAR file and the bundle is ready to be installed on the server.

Administrator Applications

Installing and starting our sample bundle can be done easily using one of the administrator applications provided with mBedded Server 4.0. The choices include:

- **SGUI:** Visual administrator using Swing technology
- **PGUI:** Visual administrator using the ProSyst GUI library
- **HTTP:** Visual administration using any standard Web browser
- **WAP:** Visual administration using a mobile phone with WAP (Wireless Application Protocol) or a WAP emulation environment
- **Telnet:** Console administrator

By using the PGUI Administrator, installing the bundle is as easy as clicking the install button and navigating to the proper JAR file. Starting the bundle is almost as easy – click the start button. If a problem exists, a dialog box will appear and offer to provide details. Access is available to information about each currently installed bundle (see Figure 1), and certain bundles provide property editors so they can be easily modified. For example, the server port for the PMP bundle could be changed here.

In addition, there's an event viewer that documents activities on the server, including when bundles were installed or services registered.

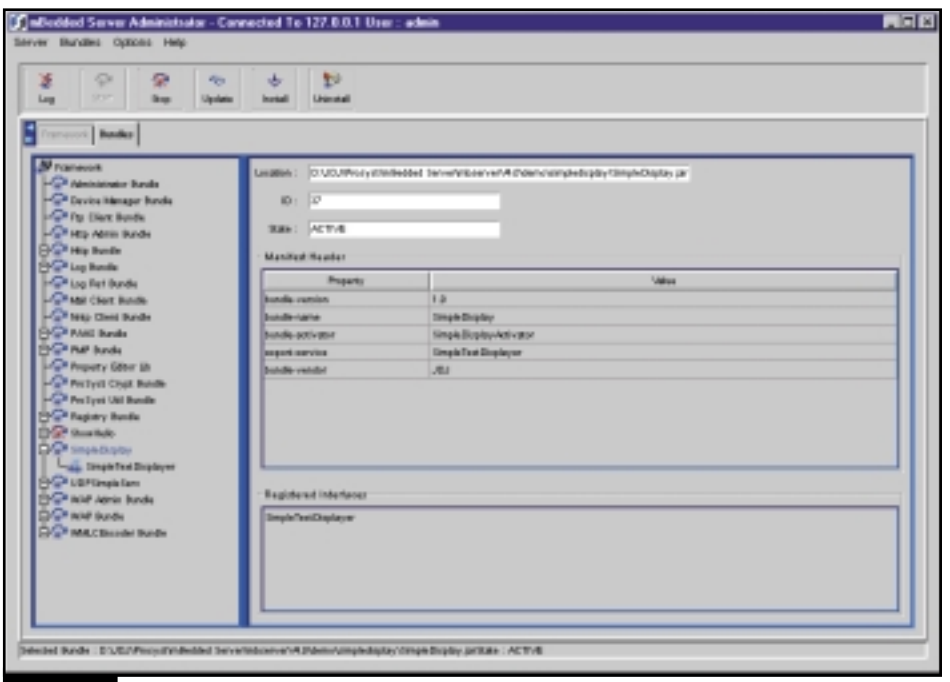


FIGURE 1 ProSyst's PGUI Administrator application

Using Services from an Application

To complete our simple demonstration, we need an application that can make a connection to the server and locate and use the SimpleTextDisplayer service (see Listing 3).

The call to connect() tries to establish a connection with the ProSyst Message Protocol (PMP) Service at port 1449, which is where the service listens and getReference() returns a registered service. The functions getMethod() and invoke() result in the execution of the service and the display of the SimpleDisplayImpl frame with the message "This String came from SimpleSend" (see Figure 2) confirming that the SimpleSend application was able to locate and execute the desired service.



FIGURE 2 Result of the TextDisplayer service

Demos and Documentation

mBedded Server 4.0 comes with numerous demos and extensive documentation. The documentation includes the APIs for the OSGi specification and for the services provided by ProSyst. ●

Listing 1

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

public class SimpleDisplayImpl extends Frame implements SimpleTextDisplayer {
    Label theTextToDisplay = new Label("",Label.CENTER);
```

```
public SimpleDisplayImpl() {
    super("SimpleDisplayImpl");
    initializeUI();
    setVisible(false);
}

public void displaySimpleText(String theNewText) {
```

```

theTextToDisplay.setText(theNewText);
setVisible(true);
}

private void initializeUI(){
setTitle("SimpleDisplayImpl");
setBounds(20,20,300,300);
add(theTextToDisplay);

addWindowListener(new WindowAdapter(){
public void windowClosing(java.awt.event.WindowEvent e) {
SimpleDisplayImpl.this.dispose();
return;
}});
}
}

```

Listing 2

```

import org.osgi.framework.*;
import java.util.*;

// Class SimpleDisplayActivator
// activates the SimpleDisplay Service bundle
public class SimpleDisplayActivator implements BundleActivator
{
public void start(BundleContext bc) throws BundleException {
try {
SimpleDisplayImpl tImpl = new SimpleDisplayImpl();
Hashtable dict = new Hashtable();

dict.put("Description", "SimpleTextDisplayer service");
ServiceRegistration servReg
= bc.registerService("SimpleTextDisplayer", tImpl, dict);
} catch (Exception e) {
System.out.println("Exception.." + e);
throw new BundleException("Failure in start method, " +
e.getMessage(), e);
}
}

public void stop(BundleContext bc) throws BundleException {

```

```

}
}

```

Listing 3

```

import com.prosyst.mbs.client.pmp.*;

public class SimpleSend {
static String SIMPLEDISPLAY = "SimpleTextDisplayer";

// This demonstrates how to
// call a method of some service in the framework through PMP.
public static void main(String [] args) {
try {
// Init connection object.
Connection con = new Connection();
con.connect("127.0.0.1",
1449, "admin", "admin", (byte) 0);

// Get remote reference to service.
RemoteObject rObject = con.getReference(SIMPLEDISPLAY, "");

String [] str = new String[1];
str[0] = new String("java.lang.String");

// Get method of service.
RemoteMethod rMethod = rObject.getMethod("displaySimple-
Text", str);
Object [] objArr = new Object[1];
objArr[0] = new String("This String came from SimpleSend");

// Call method.
rMethod.invoke(objArr, false);
System.exit(1);
} catch (Exception e) {
e.printStackTrace();
}
}
}

```



INT

International Conference for the Largest Independent Java

Santa Clara, CA – The International Conference for Java Technology – JC2 – attracted more than 2,500 attendees, representing 25 countries from around the world. It was the largest gathering of Java developers since JavaOne. The event, which took place September 24–27, featured four days of technical sessions and a two-day interactive exhibition. SYS-CON MEDIA, JDJ, and Camelot, sponsors of the event, hosted recognized Java experts, developers, leading product vendors, and IT professionals.

Delegates had the opportunity to master new skills from an exceptional lineup of educational tracks designed for users of all levels, with special sessions dedicated solely to advanced Java gurus. More than 60 sessions were offered, with over 100 hours of instruction. Presentations covered the latest developments in Java and related technologies such as XML, Web development, wireless devices, and databases.

Joshua Duhl, technical chair, says he received a lot of positive feedback: “Developers found that the breadth, depth, and degree of technical coverage was very strong, and in most cases exactly what they were looking for.” In addition, Duhl said the exhibitors were pleased with the traffic, and reported that the quality of leads and caliber of the people they were talking to was high.

The bustling exhibit floor was filled with more than 50 vendors, providing attendees with the opportunity to preview the latest Java tools. “JC2 was a huge draw for Ejjip.net,” said Dave Johnson, the company’s CEO. “Our booth was full of Java professionals who

were sincerely interested in learning about our service offerings in Java. Additionally, our team learned a great deal at some of the presentations at the conference. Ejjip.net will definitely attend next year.”

According to Ajit Sagar, editor-in-chief of *XML-J* and a speaker at the conference, “The atmosphere at JC2 was very reassuring in that the Java community has serious implementers in the market who are addressing complex issues like distributed transactions, Enterprise JavaBean usage issues, and B2B marketplaces. All in all, there was less hype and more food for thought for Java developers. I also saw representation from several companies I had never heard of, testimony to the fact that SYS-CON and Camelot are reaching Java communities around the globe.”

Jason Westra, CTO of Verge Technologies Group and EJB Home columnist, concurred: “JC2 proved to be a significant event for the Java community. There were exhibits by major industry players like Computer Associates and Borland, all showing major commit-

Participation Exceeds the Annual Java Business Internet Expo

ments to Java. The show also excelled as a forum for smaller vendors like the Middleware Company and Ampersand Corporation, marketing Enterprise Java services and new Java products. Presentations discussed everything from EJB to the J2ME and provided real-world experiences for the audience. Overall, the show represented an extremely dedicated and professional cross-section of Java professionals. A big thanks to *JDJ* for organizing the event.”

Plans are already underway to present the International Conference for Java Technology in 2001. Check out our Web site for more details!

Featured JC2 Speakers

Several of the top Java technology professionals spoke at JC2. Among them were:

George Paolini, vice president of Java Community Development for Sun Microsystems,

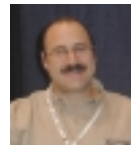


opened the International Conference for Java Technology with “The Java Community: All for One and One for All.”

Bruce Scott, CEO of Point-Base, talked about the Digital Big Bang and the Cosmology of the Data Universe. He said that dealing with the multitiered explosion of corporate and personal data requires platform-independent solutions that can span networks of servers, desktops, laptops, and mobile/wireless devices.



Paul Lipton, associate director of Object Technology, the leading vendor of Enterprise e-business solutions, shared his view on the state of Java today, and where it needs to go tomorrow.



Rod Smith, vice president of Java at IBM, presented an address entitled “IBM at Your Service:



Java Technology Becomes Event in the World!



Leading the E-Business Evolution."

M.A. Sridhar, chief architect of Ampersand, announced a new Web application design paradigm in his address, "ZeroCode: The Fastest Way to a Database Accessible Web Site." The ZeroCode development environment gives an instant 100% Java three-tier application, typically saving 80% of coding time.

A Special Address by Sean Rhody

In addition to his role as editor-in-chief at *Java Developer's Journal*, Sean is a principal consultant with Computer Sciences Corporation. In his closing keynote for the *JDJ* plenary, he

related his experiences over the past several years in developing a number of sites for e-commerce and B2B collaboration, and



addressed the Internet and its effect on reducing the time frames of modern development projects.

Sean Rhody joined Jason Westra and Jishnu Mitra from Borland in a special focus on "Real-World EJB – with the Masters."

JDJ Honors Readers' Choice Awards Winners

Sean Rhody presented the winners of the Readers' Choice Awards with the "Oscars of the Software Industry" at the International Conference for Java Technology. This year a record 20,000 *JDJ* readers voted for their favorite Java products and services in 17 categories. There were 68 winners and finalists in all. 🌐

More Than 75 Participating Vendors by Invitation Only

- 8x8 www.8x8.com
- Advanced Software Technologies www.gdpro.com
- Ampersand Corporation www.ampercorp.com
- AppStream www.appstream.com/home.html
- Ariba www.ariba.com
- Baltimore Technologies www.baltimore.com
- BusinessWire www.businesswire.com
- CFDJ www.sys-con.com/coldfusion/newcfjdj.cfm
- Concrete Incorporated www.concretemedia.com
- Computer Associates www.ca.com
- Compuware www.compuware.com/numega
- Cysive www.cysive.com
- EarthWeb www.earthweb.com
- Eliad Technologies www.eliad.com
- Emerging www.emerging.com
- Fawcette Technical Publications www.fawcette.com
- GemStone www.gemstone.com
- IAM www.iamx.com
- IBM www.ibm.com/developer/xml
- Idea Integration www.idea.com
- Informix www.informix.com
- Inprise/Borland www.inprise.com
- Intuitive Systems www.intuisys.com
- iRise www.irise.com
- Lernaut & Hauspie www.lhsl.com
- Java Developer's Journal www.sys-con.com/java/newjava.cfm
- JavaPro www.java-pro.com
- JustJavaJobs.com www.justjavajobs.com
- LearningPatterns.com www.learningpatterns.com
- Learning Tree International www.learningtree.com
- Middleware Company www.middleware-company.com
- Netergy Networks www.netergy.net
- Net-Strike www.net-strike.net
- Objectivity www.objectivity.com
- ObjectSpace www.objectspace.com
- Objecttools www.objecttools.com
- O'Reilly Network www.oreillynet.com
- Parasoft www.parasoft.com
- Pingtel www.pingtel.com
- POET Software www.poet.com
- Pointbase www.pointbase.com
- Programmer's Paradise www.pparadise.com
- RadView Software www.radview.com
- RCI Media Group www.rcimedia.com
- SDTimes www.sdtimes.com
- Segue Software www.segue.com
- SmartEnergy www.smartenergy.com
- Software Markets www.softwaremarkets.com
- StarBase Corporation www.starbase.com
- Sun Microsystems www.sun.com/
- Sun Educational Services www.sun.com/
- SurfSoft www.surfsoft.com
- SYS-CON Radio www.sys-con.com/java/javaone2000/javaone.cfm
- Tidestone www.tidestone.com
- TogetherSoft www.togethersoft.com
- TradeShowNews.com www.tradeshownews.com
- Unify Corporation www.unifywvave.com
- Verge Corporation www.vergecorp.com
- VisiComp www.visicomp.com
- Wireless www.sys-con.com/wireless
- Wrox Press www.wrox.com
- XML-Journal www.sys-con.com/xml

Developers found that the breadth, depth, and degree of technical coverage was very strong, and in most cases exactly what they were looking for

—continued from page 100

Q: What can't you get out of the code?

A: With Java, one-to-many relationships are harder to pull out of the code. The problem here is that Java collections, like Vector and Hashtable, are not typed. Because of this, most UML design tools do not identify the one-to-many relationships in your software, and force the user to manually go through all of their classes and mark the relationships accordingly. WebGain StructureBuilder, on the other hand, is the only tool that we know of that does identify the one-to-many relationship right from the code, saving the user countless amounts of time in not having to identify all those relationships manually. It just reads in your classes, and shows you the one-to-many relationships instantly.

Q: What are some of the pitfalls of code generated by modeling tools and how do you avoid them?

A: There are two major problems with code generated by modeling tools.

The first one is that most of the time the code generated is just structural stubs – it actually doesn't do very much. Now this is better than having to write this code by hand. There is much more that can be done. WebGain StructureBuilder, for

example, will allow the user to quickly generate working code and code patterns with just a click of the mouse. Everything from constructors, singleton patterns, setters, getters, equals and compares routines, to any customized templates that the user wishes to add.

The second, and far bigger, problem of code generated from modeling tools is that the code quickly becomes out of sync with the model. Changing the code after generation results in a mismatch of information between the code and the model, and synchronizing that information can be a time-consuming and painful experience for the user, which is why many users choose not to go back to the model after the code has been generated. WebGain StructureBuilder solves this problem by using the source code as its base for the model information. Therefore there is no synchronization. If you change the code outside of the product, the next time you bring up the diagrams, the information is always up to date. Since it's all one data source, no synchronization is necessary.

Q: So if I model something in a traditional UML modeling tool, export the code generated, and start working on it, will these tools be able to read it back in and recognize it?

A: Many of them will read the code back

in – there's a synchronization process. And if you changed too much in the code, then they will flag you and you have to go in manually and tell it what you did. You can generate C++ from the model data or Visual Basic, etc. Without being tightly tied to the target language and/or the development environment, synchronization will always be more difficult.

Q: Is there such a thing as too much modeling?

A: There can be too much modeling. At a certain point you need to start prototyping code and see how your initial designs look. That curve is different depending on the project, organization, and people.

Q: What's the best way to introduce a modeling tool to developers, especially to those who've never used it in the past?

A: We've found it's best to introduce UML to programmers a little at a time. Use it on different types of diagrams. The Class diagram has the most synergy with the actual Java software structure. So someone who's designing object-oriented software is doing designs of class diagrams whether they know it or not. The boxes show relations between objects. Any Java programmer viewing a class diagram will recognize it immediately.

Other diagrams are more abstract. State or activity diagrams really show a program or method flow, but show it at a high level, like a flowchart. Sometimes a programmer might view this and have a disconnect with the code they are writing.

Q: So what do you think will be the future of modeling for Java? What do you think will be the next major development in this area?

A: In the past, UML has been viewed as a "techie" type of language – real hard-core technology. Object guys understood it, some programmers did and some didn't, and business people definitely were scared away from it. I think you'll see more user-friendly UML. It needs to be less intimidating and used across the organization to share information and communicate between groups during the development process. It's imperative for the people who are defining the business requirements of the product to be involved at all stages to make sure that the requirements are being met as the application is being developed.

I think the next major development will be positioning UML as more usable and more approachable by everyone in the organization – not just the people designing software. ☪

I commit.
I believe.
I succeed.
I am Niku.

i am believe

Niku has a definite spirit. A spirit that derives from the promise of what we do - that comes with knowing our products are revolutionizing the way businesses automate themselves over the Internet. At Niku, we provide real solutions to real business problems. Our service systems can be found in companies like Cisco Systems, USI, Sybase and Gateway. We're determined to remain the industry standard of the end-to-end service economy. Are you ready to believe in something?

Join us in one of the following opportunities available in Redwood City or Petaluma, California or Chicago, Illinois:

TECHNICAL LEAD - 5+ years industry experience in software development and 2+ years experience as a Tech Lead using Java and C++. Also requires UML or other OO methodologies experience. Web application development or internet technologies experience desired. Job Code: DEV164.JDJ

SENIOR APPLICATION ENGINEER - 4+ years industry experience in software development and 2+ years experience as an App/Sr. App. Eng. using Java/ASP and C++. OO methodologies experience a plus. Web application development or internet technologies experience desired. Job Code: DEV166.JDJ

Learn more about what we do at Niku and find out about career opportunities by visiting our corporate web site www.niku.com. Internet keyword: Niku Careers. Apply to: careers@niku.com or FAX: (650) 298-3129. EOE.

NIKU

**Chicago, IL
Bay Area, CA**

Moore
Technology Resources

Moore Technology Resources is a provider of Information Technology consulting and staffing solutions to organizations ranging from e-commerce start-ups to Fortune 100. We provide the highest quality of technical staffing resources on a contract or full-time basis. It's our priority to assist IT professionals in finding leading edge hi-tech positions that enable them to utilize their current skills and also learn new ones. A wide variety of full time and long term contract openings are available.

**Java Developers
Java Architects
Project Leads
Project Managers**

**\$200,000+
Potential**

**For immediate consideration,
please call or send resume to:**

**Moore Technology Resources
700 E. Diehl Rd., #180, Naperville, IL 60563
Phone: 630-577-9000
Email: mooretec@moore-tec.com**

www.moore-tec.com



Interview...with **Bill Baloglu & Billy Palmieri**

PRINCIPALS OF OBJECTFOCUS INC. AN INTERVIEW BY **DAVID JOHNSON**

JDI: *ObjectFocus is the leading staffing firm in the Silicon Valley area. Two of the principals, Billy Palmeri and Bill Baloglu, will tell us a little about their story and what differentiates them in the Java staffing community. Billy, why don't you start by giving an introduction to ObjectFocus.*

Billy: We are, as mentioned, the Java-centric contract staffing firm for Silicon Valley. We specialize exclusively in server-side, middle-tier developers. All our positions are here in San Jose up to the Redwood shores.

We alluded to working with our clients and our consultants in, I think, a rather unique way. Everything we do is on a personal basis. Every client we have, we interview; every consultant we have, we interview. We don't work through faxes; we don't work through e-mails. Of course, we're using them as tools, but we don't accept resumes off the Internet and just submit a piece of paper to a client. I think this guarantees the highest quality of matches.

Our ratio is very, very high – in the 90th percentile, simply because Bill is an

I also saw a lot of candidates come through who just weren't a good fit at all. It was a waste of everyone's time. How does the screening process that you're talking about ensure that the 90% hit rate is achieved?

Bill: As Billy mentioned, prior to staffing and recruitment, I spent 16 years as an engineer. This was primarily in the object technology area and C++. Most recently, just prior to moving to recruitment, it was in the Java area. I actually used to speak at a lot of object technology conferences.

Let me talk to you about our process in the sense of what I think is the biggest problem in the staffing industry today, as far as agencies and the middle layers are concerned. You go to your hiring manager and say, "What are your requirements?" The hiring manager says, "I need two Java guys and two server-side engineers." What does server-side mean and how do you qualify? That's really the process. If you don't understand the hiring manager's language, you don't know what he or she is looking for. "It's not just the obstacle of

skills and put them into categories. Then when we finally meet the person face to face, we go through those projects. We find out if their skills are real. We are not just looking for a bunch of keywords on a resume. or if it's just somebody who went to a Java training course and picked up these skills. We make sure all of this is in place, then we submit candidates to clients. That summarizes our technical process.

JDI: *What types of firms, on the business side, do you integrate with to fill positions? If I worked for one of these companies, how would I get in touch and what would the process be? Do I fill out some requirement forms?*

Bill: Sure, but let me elaborate on that a bit. Basically our process is like this: we have the same type of relationship on the client side that we do with the consultants or the candidates. Again, I think one of the situations in the Valley is that agencies receive by phone, fax, and e-mail what we call job orders or job requirements, such

point. Later, we can meet with them to understand their requirements and fulfill their needs.

Billy: If I could just mention a little bit more about the consultant side. One of the things we do is work heavily from referrals. Most of the people we have working with ObjectFocus right now have come to us through referrals. We feel that any candidate we get from someone who's working with us has a little more weight than someone we just pull from e-mailed résumés. This is someone who is going to bring some real weight to the table.

We released a referral program last week that I think is unique. We actually pay 10% of the gross margin as a commission on any referral. So, if someone were to work for us for a year, the referring person could actually earn over \$5,000 a year on that referral. One referral is \$5,000, two is \$10,000, three is \$15,000. We call it a commission plan. Most companies offer a couple of thousand dollars.

Bill: You may mention that this program is actually the same plan that we're offering our internal recruiters. We're treating the community that refers an individual to us as a virtual extension of our recruitment organization.

Billy: Absolutely. And I also wanted to mention that we disclose margins. Not only to our clients but to our consultants. So we have a very open policy not only with communications but with bill rates, margins, what people are earning or paying. We have a Bill of Rights for clients and consultants on our Web site, and I invite everyone to look at that.

JDI: *I think that's an excellent practice. So if you're a Java developer and you're looking for Java opportunities, check out Object-Focus; visit their Web site at www.objectfocus.com.*

AUTHOR BIO

David Johnson is CEO of Verge Technologies Group, Inc., a Boulder, Colorado, based Enterprise Java consulting and hosting firm

“The key is that you need to meet with each client to understand and capture their requirements.”

engineer. So we get people very well screened on the technical end. Because Bill is a technical guy he understands the job requirements. I've spent about four years in the recruiting area. I know the recruiting process. By the time we get to the end and bring our consultants in for an interview, we're pretty much home. I think this is a great process not only for the client but also the consultants, because they're not jerked around. And they know the job opportunities are real.

JDI: *I've actually had the opportunity to do some consulting in the Silicon Valley area, and I did see some consultants forced to leave their team when there was a mismatch of skills.*

EJB, XML, and XSL and that type of thing, but it's also that I need a good communicator because this guy is going to be mentoring as well."

From a technical angle we understand what the manager is saying. Managers might sometimes elaborate just by saying a few words, like, "I need a Java server engineer," but we understand what he means by that. When we come back to the office and start qualifying people, we're not just looking at their résumés, and going to a search engine and typing Java, XML, and XSL. We don't just get a bunch of papers and send them to the manager. We actually read those résumés; we don't just go by the summaries. We look at each project and organize those

as requests for two Java guys with XML and XSL. The key is that you need to meet with each client to understand and capture their requirements.

I don't want to digress too much from the question you asked, but I want to elaborate on the importance of the relationship on the client side as well. We work with companies I'd like to characterize as emerging growth companies. Our clients include large companies, such as HP E-speak, Sun Microsystems, 3Comm, and Cisco as well as some of the most impressive dot-coms in Silicon Valley, e-commerce companies typically in the B2B arena.

People can get in touch with us at www.objectfocus.com. It's a good starting