

No. 1 *i*-Technology Magazine in the World

JDJ

WWW.SYS-CON.COM/JDJ VOL.10 ISSUE:5

Message exchange between a
Java client and server communicating over SSL

PAGE 42

UNDERSTANDING JSSE

RETAILERS PLEASE DISPLAY
UNTIL JULY 31, 2005

\$5.99US \$6.99CAN



06>

0 09281 01751 6

PLUS...

- ▶ JMS WS Meets the J2EE Connector Architecture
- ▶ Coding Business Logic in Excel
- ▶ Know Your Worst Friend, the Garbage Collector

Your potential. Our passion.™
Microsoft

Think big. Then build.

With Visual Studio®.NET 2003, you can build enterprise Web applications with less code, so you can turn that big idea into reality faster than you ever thought possible. For example, RAD-style Web Forms let you quickly build applications for any browser or platform. Plus, the enhanced HTML editor gives you IntelliSense® statement completion for HTML tags. All of which means you're more productive, and more ready to take on your biggest ideas. Find out more at msdn.microsoft.com/visual

© 2004 Microsoft Corporation. All rights reserved. Microsoft, IntelliSense, Visual Studio, the Visual Studio logo, and "Your potential. Our passion." are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.


Microsoft
Visual Studio

i-Technology Blogging Begins at Home



Jeremy Geelan



DESKTOP



CORE



ENTERPRISE



HOME

Editorial Board

Desktop Java Editor: **Joe Winchester**
Core and Internals Editor: **Calvin Austin**
Contributing Editor: **Ajit Sagar**
Contributing Editor: **Yakov Fain**
Contributing Editor: **Bill Roth**
Contributing Editor: **Bill Dudney**
Contributing Editor: **Michael Yuan**
Founding Editor: **Sean Rhody**

Production

Production Consultant: **Jim Morgan**
Associate Art Director: **Tami Lima**
Executive Editor: **Nancy Valentine**
Associate Editor: **Seta Papazian**
Research Editor: **Bahadır Karuv, PhD**

Writers in This Issue

Calvin Austin, Jerason Banes, Jonathan Bruce,
Yakov Fain, Jeremy Geelan, John Gilbert,
Romain Guy, Anatoly Krivitsky, Di Li,
Alex MacLinsky, Sudhir Upadhyay, Pavel Vlasov,
Joe Winchester, Frances Zhan

To submit a proposal for an article, go to
<http://grids.sys-con.com/proposal>

Subscriptions

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

888 303-5282

201 802-3012

subscribe@sys-con.com

Cover Price: \$5.99/issue. Domestic: \$69.99/yr. (12 Issues)
Canada/Mexico: \$99.99/yr. Overseas: \$99.99/yr. (U.S. Banks or Money Orders) Back Issues: \$10/ea. International \$15/ea.

Editorial Offices

SYS-CON Media, 135 Chestnut Ridge Rd., Montvale, NJ 07645
Telephone: 201 802-3000 Fax: 201 782-9638

Java Developer's Journal (ISSN#1087-6944) is published monthly (12 times a year) for \$69.99 by SYS-CON Publications, Inc., 135 Chestnut Ridge Road, Montvale, NJ 07645. Periodicals postage rates are paid at Montvale, NJ 07645 and additional mailing offices. Postmaster: Send address changes to: Java Developer's Journal, SYS-CON Publications, Inc., 135 Chestnut Ridge Road, Montvale, NJ 07645.

©Copyright

Copyright © 2005 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 Dorothy Gil, dorothy@sys-con.com. SYS-CON Media and SYS-CON Publications, Inc., reserve the right to revise, republish and authorize its readers to use the articles submitted for publication.

Worldwide Newsstand Distribution

Curtis Circulation Company, New Milford, NJ

For List Rental Information:

Kevin Collopy: 845 731-2684, kevin.collopy@edithroman.com
Frank Cipolla: 845 731-3832, frank.cipolla@epostdirect.com

Newsstand Distribution Consultant

Brian J. Gregory/Gregory Associates/W.R.D.S.
732 607-9941, BJGAssociates@cs.com

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.



When we opened up the *JDJ* domain (javadevelopersjournal.com) to bloggers everywhere, we knew the take-up would be good. But one thing we couldn't be certain about in advance was whether the blogs themselves would be any good. We needn't have worried.

As many of you will already have found out, the editors of *JDJ* all blog regularly, and naturally RSS feeds are available too – so if in between issues of the magazine you want to read something by, say, Ajit Sagar, all you need to do now is scoot over to <http://ajitsagar.javadevelopersjournal.com> or subscribe to Ajit's RSS feed: <http://java.sys-con.com/read/rss/9.htm>. Likewise you'll find Calvin Austin's feed at <http://java.sys-con.com/read/rss/65.htm>, and so on.

A list of RSS feeds to all the *JDJ* Editors' Blogs can be found at the new *JDJ* main page (<http://jdj.sys-con.com>), where you'll always be just one click away from being able keep up with the latest thoughts of Ajit and Calvin – or those of Joe Winchester, Yakov Fain, Bill Dudney, and Michael Yuan. Jason Bell (<http://java.sys-con.com/read/rss/66.htm>) is back now on the *JDJ* team, too.

Talking of Michael Yuan, he is going to run a JavaOne BoF next month titled "Smarter Rich Client Through Middleware Services." Given the recent interest in running Swing apps inside a container, he's expecting developers with interesting ideas on middleware services that can benefit Swing apps to come to the BoF and share them (or you can e-mail him at michaelyuan@sys-con.com). Michael will also, he says, be discussing more "traditional" client-side container topics such as the OSGi.

Now that he has left Sun and joined the Kim Polese start-up, SpikeSource, Calvin can at last blog source code. Though talking of source code, perhaps you prefer to keep in touch with the thoughts and views not of our esteemed editors but of regular *JDJ* readers like Uday Kumar (<http://uday-kumar.javadevelopersjournal.com/>), who describes his blog as the "musings of a software philosopher"). Most recently – for

example – you'd have been able to read Uday's commentary on the source code of the Jasper compiler.

Before that Uday blogged a fascinating discussion on the CLR (in .NET) in comparison to the JVM. "In the JVM world," he observed, "the Java code is translated into byte code by a compiler, after which the byte codes are interpreted and executed by the JVM. Compilation of byte code into native CPU instructions is optional. The CLR, however, is more than a virtual machine; in fact the CLR is more akin to the J2EE container than the VM. In the CLR world, the programs written in a host of languages (C#, VB.NET, Java, etc.) are converted to a byte code by the respective compilers. These are compilers that target the CLR: in that sense the CLR is like a VM – it abstracts away the underlying CPU."

One more quick example of a reader blog worth subscribing to is <http://straxus.javadevelopersjournal.com/> – where a recent entry discussed Sun and open source. "It's interesting to see the extent to which Sun has adopted Open Source ideas and principles, at least in part," blogged Straxus. "On the one hand, they've released several major pieces of software into the Open Source world, and opened up a boatload of patents to Solaris developers. On the other hand, Java is not Open Source, and shows no signs of changing that status any time soon."

"There seems to be a certain amount of duality in Sun's public activities," Straxus concludes, "and I suspect that this might be part of an internal corporate struggle to determine which way (proprietary or open) is the best way to go. It'll be interesting to see how this shakes out over the next year or so."

It will be interesting too to see how blogging at javadevelopersjournal.com shakes out. The early signs are that it is going to be a fast-growing community, and to encourage its widening and deepening we are already devoting space online to featuring blogs that inform, stimulate, entertain, or inspire. So if you haven't started one yet, now's the time; after all, who's to say it won't be your latest blog that's featured one day in an editorial here in *JDJ*? ☛

Jeremy Geelan is group publisher of SYS-CON Media and is responsible for the development of new titles and technology portals for the firm. He regularly represents SYS-CON at conferences and trade shows, speaking to technology audiences both in North America and overseas.

jeremy@sys-con.com

Focus on results

Set your sights on StyleVision® 2005,
the ultimate e-forms, DB report and
stylesheet designer from Altova®.

Presented in Version 2005:

- XSLT 2.0 and XPath 2.0 support
- Output to Word/RTF format
- Creation of database-enabled electronic forms

Scope out StyleVision 2005 and see a marked
improvement in your data presentation capabilities.

StyleVision is the only application for transforming
XML or database content into HTML, PDF and
Word/RTF output all from the same design. In one
simple step you can access your data, process it,
and render it into intuitive Authentic® electronic
forms and eye-catching HTML pages, PDF
reports and Word/RTF documents.

Get more from your designs!

**Download stylevision® 2005
today: www.altova.com**

JDJ contents

JDJ Cover Story

42

Understanding JSSE

by Sudhir Upadhyay

*Message exchange between a Java client
and server communicating over SSL*

FROM THE GROUP PUBLISHER

z-Technology Blogging Begins at Home

by Jeremy Geelan

3

VIEWPOINT

SOA and XQuery

by Jonathan Bruce

6

JAVA ENTERPRISE VIEWPOINT

Another Brick in the Wall

by Yakov Fain

8

DESIGN PATTERNS

Designing Custom Multithreading Frameworks in J2EE Containers

A way to improve the performance of Java apps

by Di Li

10

WEB APPLICATIONS

Remote Benchmarking with Servlets and JSF

Wringing out performance

by Anatoly Krivitsky

16

ENTERPRISE

Three Sources of a Solid Object-Oriented Design

*Design heuristics, scientifically proven OO
design guidelines, and the world beyond
the beginning*

by Gene Shadrin

24

CDN

Building Web Apps That Leverage Content Delivery Networks

Five different integration options

by Alex Maclinovsky

28

API

Coding Business Logic in Excel

Morphing an Excel spreadsheet into

Java business logic

by Jerason Banes

34

SERVICES

AOP-Enabled ESB

Updating the enterprise nervous system

by John Gilbert

36

CORE AND INTERNALS VIEWPOINT

What's in a Name: Is This the End of J2EE?

by Calvin Austin

40

DESKTOP JAVA VIEWPOINT

Total Eclipse

by Joe Winchester

48

DATABASE

SQL Compiler (for Java)

*Free your Java code from SQL statements –
compile them to Java classes*

by Pavel Vlasov

50

PRESSROOM

Industry News

JDJ News Desk

62

Features

20



JMS Web Services Meet the J2EE Connector Architecture

by Frances Zhao

58



Know Your Worst Friend, the Garbage Collector

by Romain Guy



Jonathan Bruce

SOA and XQuery

The rise in prominence of Service Oriented Architectures (SOA) has triggered a storm of debates on how best to build enterprise SOA-based applications and establish a predominant industry position. In the meantime, the W3C XQuery language is accelerating toward "Recommendation" status; although comparatively little time has been spent debating XQuery's role in SOA, I expect that its rising prominence will soon capture the attention of many Java developers seeking to build enterprise SOA-based applications.

The W3C XQuery specification provides a native XML query language that integration platforms and components can use to aggregate data under the unifying umbrella of XML. XQuery levels the data integration playing field by providing a single interface that lets developers access multiple data sources under a unifying data model.

Why is XQuery so important? In talking with many customers building SOA applications,

I have found that one of their greatest challenges lies with data integration. Business-critical data is typically stored in relational database management systems (RDBMSs), providing reliable and centrally managed data repositories. In spite of the dominant position of relational data, the growth of XML (especially as a format for exchanging data over the Internet) has forced business applications to function seamlessly with both XML and relational data.

Fortunately, W3C XQuery has emerged as an indispensable approach for integrating data, in particular when working with XML and relational data. To that end, I believe developers will increasingly rely on XQuery technology that has the ability to abstract any data source as XML, lets them work with XML and

relational data together in a highly optimized fashion, and reduces the effort of building applications suited to the demands of SOA.

For example, we are already seeing the major RDBMS vendors embedding XQuery support as a means to exposing relational data as XML data sources, therefore implicitly increasing data portability and accessibility via XQuery. RDBMSs without integrated support for XQuery will continue to delegate the responsibility to the middle tier to ensure their equal participation in increased data integration. In fact, I believe it is likely that the middle tier will emerge as the sweet spot for Java developers to es-

tablish an integration component end-point (ICE) as a means to integrate a set of distributed data sources. In this scenario, technologies like DataDirect XQuery from DataDirect Technologies will let developers weave together distributed relational data sources with XML data and expedite the migration toward SOA-based applications.

For those groaning at the prospect of yet another query language, the power

of XQuery and the productivity gains of using XQuery for data aggregation amplify how XQuery can work in concert with business applications evolving toward SOA. Significantly, XML IDEs already provide extensive tools to help developers build and migrate their Java applications to use XQuery, thus maximizing their business data for SOA.

XQuery uniquely safeguards corporate investment in relational databases, while enabling established data for participating in your enterprise's SOA push. As the power of the XQuery language becomes more fully appreciated, I expect the importance of middle-tier XQuery implementations will result in significant productivity gains for Java developers building and deploying applications. ●



Jonathan Bruce

is XQuery Technology Evangelist for DataDirect Technologies. He joined DataDirect after seven years at Sun Microsystems where he served as the JDBC Specification Lead and Architect for the Java platform. Jonathan is co-author of the book *JDBC API Tutorial and Reference, Third Edition*, holds a number of patents, and frequently speaks at industry conferences. He holds a bachelor's degree in computer engineering and a master's degree in mathematics from the University of Dublin's Trinity College in Ireland.

jonathan.bruce@
datadirect.com

President and CEO:

Fuat Kircaali fuat@sys-con.com

Vice President, Business Development:

Grisha Davida grisha@sys-con.com

Group Publisher:

Jeremy Geelan jeremy@sys-con.com

Advertising

Senior Vice President, Sales and Marketing:

Carmen Gonzalez carmen@sys-con.com

Vice President, Sales and Marketing:

Miles Silverman miles@sys-con.com

Advertising Sales Director:

Robyn Forma robyn@sys-con.com

National Sales and Marketing Manager:

Dennis Leavey dennis@sys-con.com

Advertising Sales Managers:

Megan Mussa megan@sys-con.com

Kristin Kuhnle kristin@sys-con.com

Associate Sales Managers:

Dorothy Gil dorothy@sys-con.com

Kim Hughes kim@sys-con.com

Editorial

Executive Editor:

Nancy Valentine nancy@sys-con.com

Associate Editor:

Seta Papazian seta@sys-con.com

Production

Production Consultant:

Jim Morgan jim@sys-con.com

Lead Designer:

Tami Lima tami@sys-con.com

Art Director:

Alex Botero alex@sys-con.com

Associate Art Directors:

Abraham Addo abraham@sys-con.com

Louis F. Cuffari louis@sys-con.com

Assistant Art Director:

Andrea Boden andrea@sys-con.com

Video Production:

Frank Moricco frank@sys-con.com

Web Services

Information Systems Consultant:

Robert Diamond robert@sys-con.com

Web Designers:

Stephen Kilmurray stephen@sys-con.com

Percy Yip percy@sys-con.com

Accounting

Financial Analyst:

Joan LaRose joan@sys-con.com

Accounts Payable:

Betty White betty@sys-con.com

Accounts Receivable:

Gail Naples gailn@sys-con.com

SYS-CON Events

President, SYS-CON Events:

Grisha Davida grisha@sys-con.com

National Sales Manager:

Jim Hanchrow jimh@sys-con.com

Customer Relations

Circulation Service Coordinators:

Edna Earle Russell edna@sys-con.com

Linda Lipton linda@sys-con.com

JDJ Store Manager:

Brunilda Staropoli bruni@sys-con.com

ELIMINATE PERFORMANCE ISSUES AND RELATED ANXIETY.

**FOR RELIEF OF STRESS DUE TO SUBOPTIMAL PERFORMANCE
JBuilder IS PROVEN EFFECTIVE ACROSS THE APPLICATION LIFECYCLE.**

JBuilder® from Borland® relieves the stress of performance anxiety associated with Application Development Dysfunction. A powerful component of Software Delivery Optimization (SDO) from Borland, JBuilder rapidly improves* individual performance and productivity. But it doesn't stop there. Thanks to deep integration across the entire Application Lifecycle, JBuilder can help your entire team create better software, faster. Don't let performance issues keep you from success.

Borland®



ASK BORLAND IF SDO IS RIGHT FOR YOU. borland.com/jbuilder

* JBuilder is indicated for organizations that wish to attain real competitive advantage in the marketplace. Side effects include increased margins, greater customer satisfaction and improved efficiency.
© 2005 Borland Software Corporation. All rights reserved. All Borland brand names are trademarks or registered trademarks of Borland Software Corporation in the United States and other countries. 23389



Yakov Fain
Contributing Editor

Another Brick in the Wall

Do you feel that being a Java guru sets you apart and makes you indispensable in your company? Or are you an entry-level person scared of being laid off given all these outsourcing trends? What are your career choices in the corporate world? Put on your headphones, turn on Pink Floyd's album *The Wall*, and let's talk...

Programmers earn their living by working either as employees or as temporary contractors. Often, people use the term *consultant* when they're referring to the employment status of a person, but this is just not right, because the word *consultant* means a subject expert, while the word *contractor* means a temporary worker and a separate legal entity, which is exactly what consultants are. There is an opinion that permanent employment provides better job security, but let's take a closer look at two former college roommates, Alex and Steve, who graduated from the same college eight years ago.

Alex was always dreaming of being an employee of a large corporation. He knew that he'd be more secure there (*Momma's gonna keep baby cozy and warm*) and was ready to work for such a firm for many years. He found such a job and had to start from scratch learning the rules of the corporate world: your phone conversations may be recorded, a designated person will browse your e-mails, your applications will be protected by a couple of firewalls and DMZ (*Momma won't let anyone dirty get through*). He had been promised a yearly training and planned to visit San Francisco while studying new Java technologies at the JavaOne conference... Sorry, but our training budget is not as good as it used to be (*We don't need no education*), but we have an exciting Six Sigma training coming up, which will greatly help your career, and you may even earn a green belt in a couple of years. He learned to play politics, and got used to working late hours to meet the unrealistic deadlines that were set by some incognito bad

person from up above. Alex met all deadlines because bonus time was looming ahead (*If you don't eat yer meat, you can't have any pudding*).

Steve decided to work for himself, so he opened a one-man company and started his career as a contractor. Even though his contracts were usually long term, Steve always knew that he needed to maintain good technical skills to be prepared for the next technical interview. He was the first to learn Aspect Oriented Programming, SOA principles, and all possible Java application frameworks that have implemented the MVC design pattern. Steve was always the only person in the building who knew exactly what the garbage collector did to the young generation. He never complained if his next client was several thousand miles away from his hometown (*Daddy's flown across the ocean leaving just a memory*).

About three years ago, by pure coincidence, Steve got a project with the same company and division where Alex has been working all these years. He was one of hundreds vice presidents with a six-figure salary, wearing an expensive suit, Six Sigma brown belt, and matching shoes. The friends were happy to work with each other, but this did not last long. The firm decided to lay off several hundred of employees and let go of most of the contractors. Alex was too expensive for the firm and Steve's contract

ended sooner than expected (*All in all you were all just bricks in the wall*). Alex received a decent severance package that allowed him to spend the next six months brushing up his Java skills and this kept paying the bills. Steve did not get any compensation but found a new gig pretty quickly in two months.

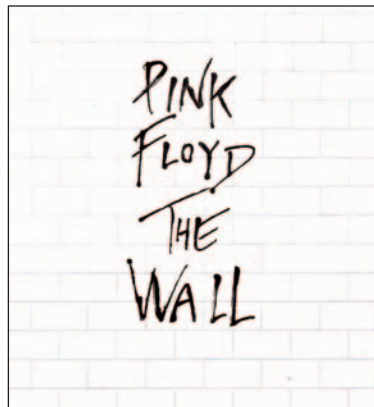
So what's the moral of this story?

If you're young and ambitious, spend at least some time working as a contractor. Do not be afraid to start fresh every now and then; this is what capitalism is all about. Besides, the average length of full-time employment of young programmers is also not more than two to four years. As you get older (over 50 in the U.S.), you'll experience difficulties in finding pure programmer's jobs (*Hey you! Out there in the cold getting lonely, getting old, can you feel me*); however, I do know a mainframe contract programmer who turns 70 this month (happy birthday, Felix!). Of course, he can't write as many "if-else" statements per minute as a college graduate, but he knows his application inside out, and the firm is not planning to get rid of him.

If you prefer full-time employment, be loyal to the company you work for. The firm's interests should take priority over your personal goals, but don't get lazy. Keep your technical skills up to date; read professional books and magazines; and visit Java online forums on a regular basis. During difficult times your employer will let you go without thinking twice: this is also what capitalism is about. Gurus will have to go because their salaries are too high, and junior developers will be replaced by an inexpensive workforce overseas. But this is okay as long as you are technically sound, have a positive attitude toward life, and accept that *all in all you were all just bricks in the wall*. ☺

References

- Pink Floyd, The Wall Album
http://www.pinkfloyd-co.net/disco/wall/wall_album.html



Yakov Fain is a Java 2 Enthusiast and Educator from Wall Street.

He is the author of the best selling book, *The Java Tutorial for the Real World*, an e-book Java Programming for Kids, Parents and Grandparents. Fain also authored several chapters for *Java 2 Enterprise Edition 1.4 Bible*. On Sundays Yakov teaches Java (see www.smart-dataprocessing.com)

yakovfain@sys-con.com

Keep Your Skills Ahead of the Crowd

Keeping your IT skills ahead of the crowd is not as difficult as most people fear. Staying on top of the trends may seem like a daunting task if, like most people, you assume that each new technology is a completely new invention that you must learn from the ground up. Fortunately, nothing is really all that new. Inventors typically create new technologies by studying existing technologies, then building upon them in ways that extend and improve them. 100% new technological advancements are very rare.

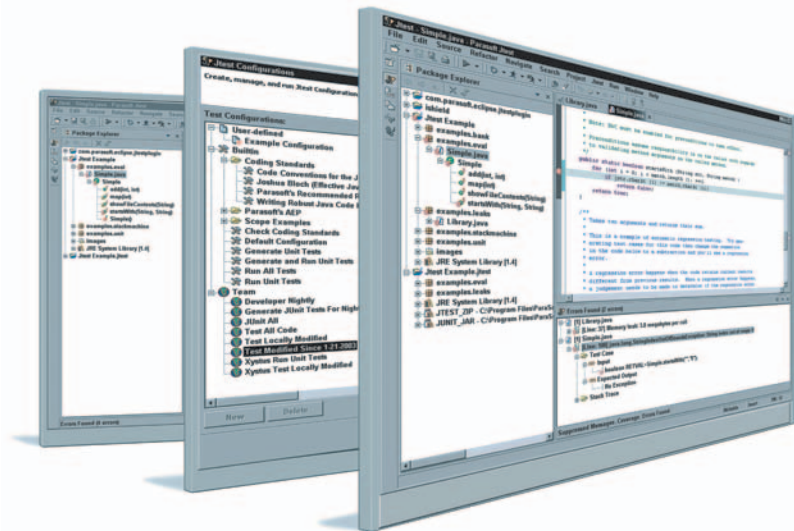
Inventors almost always leverage legacy technologies as they invent new ones. Why not leverage your own knowledge of those legacy technologies as you try to learn about the new inventions? To learn about new technologies as painlessly as possible, consider how each new advancement is similar to what you already know.

For example, consider Web services. Web services are a new trend, but — at a technological level — the parts of a Web service are not all that unique. Web services are based on remote procedural calls — messages sent to a server, which calls the requested function. RPCs were developed years ago, and are hardly a new concept. Really, the only "new" thing in Web services is the standard that is being used to write the application. If you break down Web services in this way, it's easy to learn about them. To continue with this process, you might next explore the payload requirements, the process for determining what function to call, and how the call works. As you can imagine, it's a lot more efficient — and interesting — to learn about a new technology based on its relation to familiar technologies than to learn about it by reading the specification cover to cover.

As always, the devil is in the details. But most details are critical only if you want to specialize in a given technology. For instance, if you want to specialize in Web services, you need to familiarize yourself with the details of Web service development. In that case, your next step would be to learn how to format the messages, how to expose Web services, and so on.

— Adam Kolawa, Ph.D.
Chairman/CEO of Parasoft

Deliver better Java code in less time, with fewer resources.



Parasoft® Jtest® makes Java unit testing and coding standards analysis fast and painless.

With just a click, you can verify that your Java code is robust, high quality and secure. Jtest automatically verifies compliance to hundreds of coding rules while automatically generating and executing JUnit test cases — creating harnesses, stubs and inputs.

See how Jtest can enhance your Java development efforts...

Visit our Parasoft Jtest Resource Center at www.parasoft.com/jtest_JDJ for the latest information — including white papers, webinars, presentations, and more.

For Downloads go to www.parasoft.com/jtest_JDJ



Email: jtest_JDJ@parasoft.com Call: 888-305-0041 x2174

Features	Benefits	Platforms	Contact Info
<ul style="list-style-type: none"> • Full Eclipse integration • Quick Fix automatically corrects errors • Automatically generates JUnit test cases • Customizable testing and reporting 	<ul style="list-style-type: none"> • Makes error prevention feasible and painless, which brings tremendous quality improvements, cost savings, and productivity increases • Makes unit testing and coding standard compliance feasible and painless • Encourages the team to collaborate on error prevention 	<ul style="list-style-type: none"> • Windows 2000/XP • Linux • Solaris 	<p>Parasoft Corporation 101 E. Huntington Dr., 2nd Flr. Monrovia, CA 91016 www.parasoft.com</p>

Designing Custom Multithreading Frameworks in J2EE Containers

An efficient way to improve the performance of Java apps

A custom multithreading framework is an efficient way to improve the performance of Java applications. It uses an asynchronous parallel pattern to implement the business process. However, its traditional Java thread-based implementation shouldn't be used in applications hosted in a J2EE Application Server because the threads in that framework are beyond the control of the J2EE Container. This article will discuss an intelligent solution for implementing custom multithreading by using message-driven beans. This message-driven bean based multithreading framework enables the J2EE container to manage threads and multiple MDBs to execute the business logic in parallel.

Java multithreading makes maximum use of the CPU by keeping the processor's idle time to minimum. To avoid the overhead of creating threads on-the-fly, every J2EE application server container has a default thread pool that contains a set of execute threads. The container can borrow one of the threads from the pool, and assign it a task. Upon completion of the task, the thread goes into a wait state ready to accept another assignment. The J2EE container monitors all execute threads in that pool. Custom threads are different from execute threads. They are created and controlled by the applications rather than the container. This article will introduce a solution implementing custom multithreading in a J2EE container.



Di Li, a Sun certified enterprise architect, is an expert in business modeling, enterprise architecture design, middleware architecture design and software architecting. He has 12 years of experience in software engineering and technical infrastructure. He works as lead enterprise architect for several large-scale and high-volume e-banking systems.

victor1998@hotmail.com

Asynchronous Parallel Design Pattern

The analysis model of the custom multithreading framework is based on the concept of asynchronous parallel design pattern. The pattern combines the advantages of delegate, asynchronous, and parallel design patterns. It's a model that consists of an asynchronous interface and a business engine. The business engine is composed of multiple parallel instances of business services, and fulfills the business processes. Figure 1 shows

the class diagram representing the structure of the asynchronous parallel pattern.

The client doesn't have to have knowledge of the services that the back-end business engine provides. It simply sends the requests to the delegate, which adds the incoming requests to a mediator repository and returns a result to the client. The client doesn't go into a block state waiting for a response from the back-end business engine. The repository notifies the business engine when new requests are received. The multiple parallel instances of the services in the business engine then pick up the requests from the repository and process them concurrently. Figure 2 shows sequence diagrams that illustrate typical interactions among the components of the asynchronous parallel pattern.

Our example is based on an online brokerage system. We choose to model the process of an investor placing an order that is routed to the market for execution. I simplify the business scenario so we can focus on the custom multithreading framework.

Java Thread-Based Custom Multithreading Framework (CMF)

The components of a Java thread-based CMF include a Thread Pool, Worker Thread, Task Dispatcher, Task, and Task Queue. The source code of all samples can be downloaded from www.sys-con.com/java/sourceec.cfm. The class diagram in Figure 3 shows the solution model of a Java thread-based multithreading framework.

The Task interface represents the abstract of the business cases fulfilled in the business process. Business cases are modeled by the Java class that implements the Task interface. For example, the Trade class represents the Stock Trade in a stock trading process. The business logic for processing each business case is implemented in the 'execute' method of each class.

The TaskFIFOQueue and the TaskDispatcher classes implement the asynchronous interface of the asynchronous parallel pattern. The TaskFIFOQueue class is designed as an intermediary repository that stores unsigned tasks. The TaskDispatcher class works as the delegate of the framework, which receives clients' requests, creates an instance of a Java class implementing Task interface for each incoming request (such as Trade class), adds a new instance to TaskFIFOQueue. The client can continue with other activities without having to wait for its request to be processed. The code snippet in Listing 1 at the end of this article shows how to add or get a task from the TaskFIFOQueue.

The business engine of the asynchronous parallel pattern is implemented by the WorkThread class and the ThreadPool class. The WorkThread class provides the threads that will process the assigned task. Each instance of the WorkThread class is a custom thread, created and controlled by the application. The ThreadPool class serves as the central control point to manage custom threads. Using an object of the ThreadPool class to store running threads lets us allocate custom threads in a predictable way, reuse an existing thread for multiple tasks, and save time for thread instantiation and initialization. The code in Listing 2 at the end of this article demonstrates how to implement the business engine. (Listing 2-5 can be downloaded from www.sys-con.com/java/sourceec.cfm.)

When the TaskDispatcher receives a request from a client, it creates an instance of a particular Task-type class, such as the Trade class, and adds the instance to the TaskFIFOQueue. The TaskFIFOQueue then notifies all waiting threads that a new task is ready to be picked up. The current active WorkerThread gets the task from the TaskFIFOQueue and starts to process the

task by running the `execute()` method of the particular Task-type class. After the `WorkThreads` finishes the task, it goes into a wait state and is ready to accept the next task. The sequence diagram in Figure 4 illustrates the sample collaborations for the CMF.

Although Java thread-based CMF provides an efficient solution to improve the overall performance of Java applications, it shouldn't be used in applications hosted in the J2EE application server. Java thread-based CMF is discouraged in the J2EE applications for the following reasons.

J2EE Application Server Can't Control Custom Threads

The J2EE application server container is configured to manage all the threads in the Java Virtual Machine. The threads from the CMF won't carry the implicit context that a container normally puts on a thread. The container loses control over the custom threads created explicitly by the CMF because it's not aware of their existence.

Consequently, if a custom thread fails, the container will not be able to detect the failure and so can't recover the thread – either by reclaiming the resources occupied by the particular thread or by restarting it to complete the assigned task. The presence of unmanaged failed threads in the application server can seriously impact the stability, scalability, and performance of the application server.

Thread-Based CMF Affects the Load Balancing in the J2EE Application Cluster

Since a J2EE application server isn't aware of the custom threads in the CMF, it's also not aware of the work being done by the threads, and so won't account for the impact that the additional threads have on the overall workload of the server. This will affect the load balancing of the J2EE application cluster. It will potentially over-assign work to the server, and result in significant throughput or performance problems with the server.

The EJB 2.0 specification integrates EJB with JMS, and defines a new type of Enterprise JavaBean: Message-Driven Bean (MDB). It enables the EJB container to support the asynchronous parallel design pattern. Because the basic theories of the Java thread-based CMF don't negatively impact the application server runtime, we'll use the same analysis model to design the MDB-based CMF. For the purpose of demonstration, the MDB-based framework will be implemented and deployed on WebLogic Server 8.1.

The MDB-Based Multithreading Framework

We implement the CMF using MDB because the EJB container supports concurrency and multithreading, and the MDB supports asynchronous interactions. Figure 5 depicts the conceptual view of the MDB-based CMF.

Task Interface

The Task interface in the MDB-based framework serves the same purpose as it does in the Java thread-based framework. An object of a class implementing the Task Interface represents a specific business case. The following code illustrates how to implement the Task interface:

```
public interface Task
extends java.io.Serializable {
    public void execute();
}
```

Because an EJB container is processing incoming messages in parallel threads, it's possible for a later message to be processed before an earlier one. Since each JMS message contains a Task-type object, the MDB-based framework shouldn't make assumptions about the order in which tasks are delivered to the instances of the `WorkerMDB` class. Consequently, the relationships among Task-type objects should be independent rather than sequential – each Task-type object is isolated from one another.

To remove message affinity, each client request should be designed to include the context of one business

case. For example, the context of processing one Trade in an online brokerage system includes the symbol of one stock, the cost of purchasing or selling share, the expected price, the investor's user ID and trading password. This information can be carried by multiple JMS messages or just one message. If

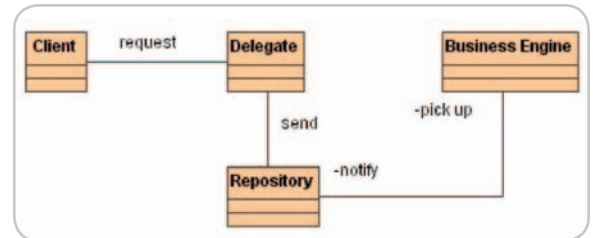


Figure 1 The structure of the asynchronous parallel pattern

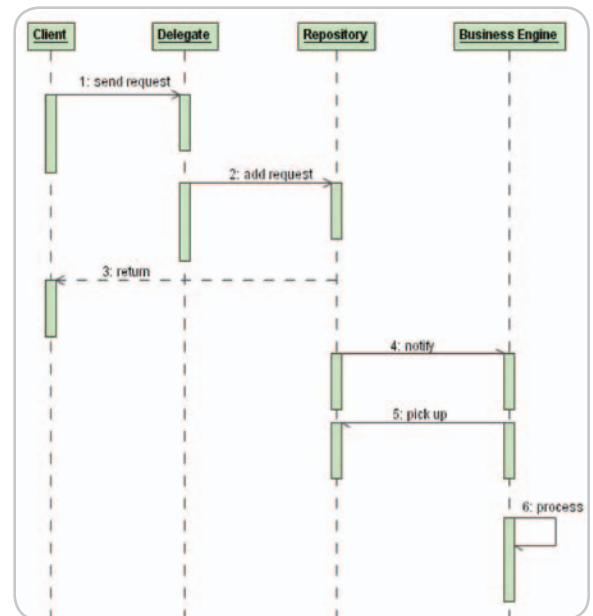


Figure 2 The typical interactions for the asynchronous parallel pattern

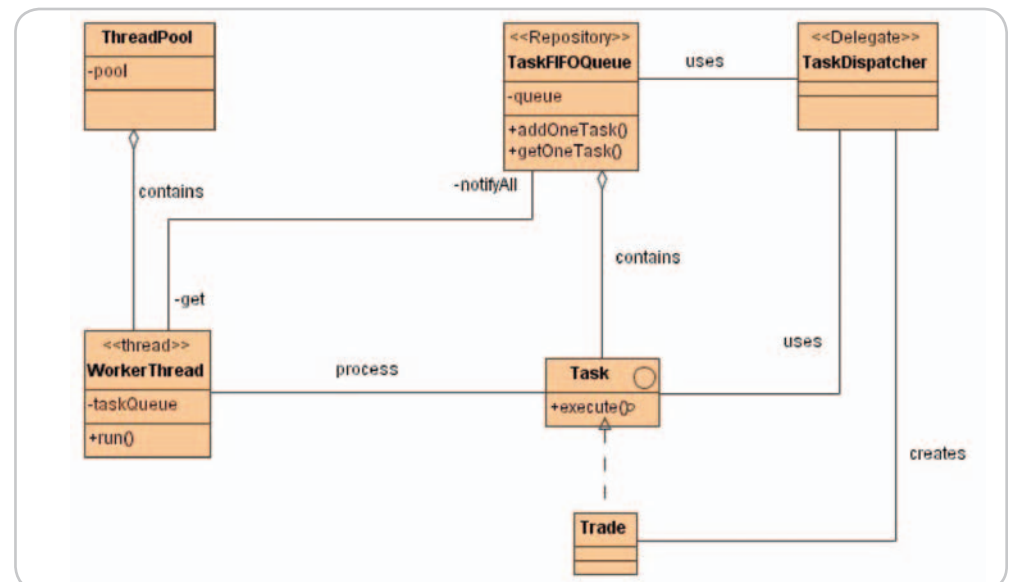


Figure 3 The solution model of a Java thread-based multithreading framework

they're encompassed in one message, the system can process each Trade by one incoming message. As a result, we remove the order of messages.

Task Queue

The Task queue replaces the role of the TaskFIFOQueue class in the Java thread-based framework. It's defined as a JMS queue in an application server or message-oriented middleware. Since each message in the Task Queue includes a Task-type object that represents a specific business case, the loss of a message in the Task Queue means that a specific business case in a business process isn't executed.

The non-persistent message is cached in memory during messaging. When the JMS provider fails, the message can't be recovered. The persistent message, on the other hand, is stored in persistent storage during messaging, and is recovered when the JMS provider restarts.

The MDB-based framework uses a persistent JMS delivery model to prevent from losing messages in case of system failure. When the server crashes, the tasks in the Task queue are stored in persistent storage, such as the DBMS or files. When the server restarts, the tasks become available again and are reassigned to the instances of the WorkerMDB class. Since each task is isolated, the delay in executing the tasks won't undermine the whole business logic. The following snippet from WebLogic Server 8.1 `configure.xml` illustrates how to create a Task queue that persists messages in an Oracle DBMS.

```
<JDBCConnectionPool
  DriverName="oracle.jdbc.OracleDriver"
  Name="CustomThreadOracleJDBCPool"
  Password="{3DES}JPF/fsddu8f7r8QAqZdURg=="
  Properties="user=weblogic"
  Targets="examplesServer"
  TestTableName="SQL SELECT 1 FROM DUAL"
  URL="jdbc:oracle:thin:@test:1521:sample"/>

<JMSJDBCStore
  ConnectionPool="CustomThreadOracleJDBCPool"
  Name="CustomThreadJMSJDBCStore"
  PrefixName="TaskStore"/>

<JMSServer
  Name="CustomThreadJMS Server"
  Store="CustomThreadJMSJDBCStore">
  <JMSQueue
    CreationTime="1104850073298"
    JNDIName="customThread.task-
      Queue" Name="TaskQueue"
    StoreEnabled="true"/>
  </JMSQueue>
</JMSServer>

<weblogic-enterprise-bean>
  <ejb-name>customThread.WorkerMDB</ejb-name>
  <message-driven-descriptor>
    <pool>
      <maxbeans-in-free-pool>
        200</max-beans-in-free-pool>
      <initial-beans-in-free-pool>
        20</initial-beansin-free-pool>
    </pool>
    <destinationjndi-name>
      customThread.taskQueue</destination-
        jndi-name>
    </message-driven-descriptor>
    <enable-call-by-reference>
      True</enable-call-by-reference>
  </weblogic-enterprise-bean>
```

JMX-Based Monitoring Framework

Java Management Extensions (JMX) define the architecture and services to actively administer resources. The JMX was originally released as individual Java technology, but now it's a part of J2SE 5.0. For demonstration purposes, we'll design and implement a JMX-based monitoring framework on WebLogic Server 8.1. The monitoring framework provides the service of capturing and notifying the status of MDB-based multithreading framework before it occurs. The component diagram in Figure 6 illustrates the architectural model of the monitoring framework.

The monitoring framework is based on a JMX notification model. It is made up of Monitor, Manager, and Listener components. The Manager component controls the start and stop of the monitoring framework. The code snippet in Listing 5 shows how to start the framework to monitor the queue "customThread.taskQueue". That's configured on multiple application servers for the MDB-based framework.

The Monitor component of the monitoring framework exposes the runtime status of the MDB-based multithreading framework. For example, when the number of messages in the Task queue exceeds the threshold, the Monitor sends a JMX notification. When the Listener component gets the notification, it starts writing data to files. Here's a sample Listener component:

```
public class MonitoringFrameworkListener
  implements RemoteNotificationListener {
  public void handleNotification(
    Notification n, Object handback){
```

SessionDispatcherBean Class

Like the TaskDispatcher class, the SessionDispatcherBean class works as the delegate of the MDB-based framework. Because the SessionDispatcherBean class is implemented as a stateless Session Bean, and the MDB-based framework can be deployed on multiple application servers in the cluster, a failure of one application server, won't affect the service of custom multithreading. The code in Listing 3 illustrates how the SessionDispatcherBean class handles a client's request of processing a trade.

WorkerMDB Class

The WorkerMDB class plays the same role as the WorkerThread class in the Java thread-based framework. It implements the function of the custom thread. The WorkerMDB class is a Message-Driven Bean. The code in Listing 4 shows how the WorkerMDB class processes the task when it gets the message from the Task Queue.

EJB Container

The EJB container in the MDB-based framework performs the role of the ThreadPool class in the Java thread-based CME. The WorkerMDB instances provide the multithreading service, and the EJB container manages the lifecycle of the WorkerMDB instances. The following snippet from WebLogic Server 8.1 `weblogic-ejb.jar.xml` is a sample of configuring an MDB pool.

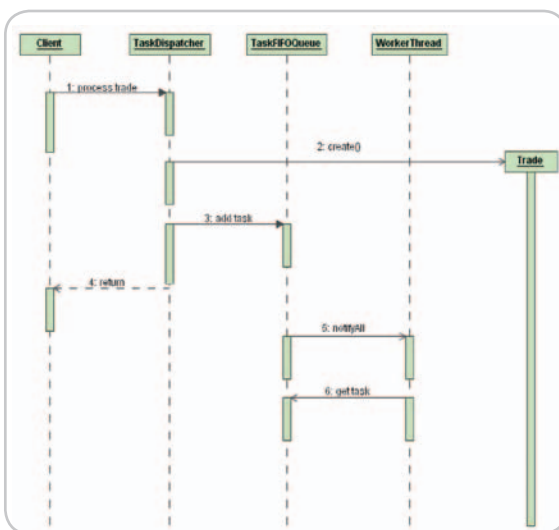


Figure 4 Sample collaborations for the Java thread-based multithreading framework

if (wantToGoHomeSooner)



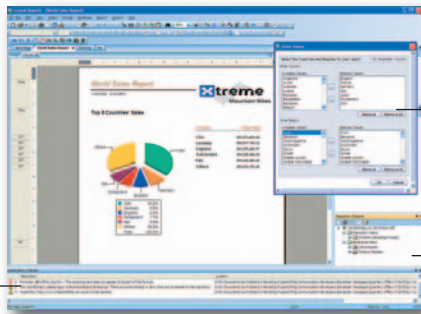
NEW Crystal Reports XI helps you spend less time integrating reports.

Visual Report Designer

Use an updated point-and-click designer to create reports and alleviate intensive coding.

NEW Dynamic Cascading Prompts

Minimize report maintenance with automatically updated pick lists and cascading prompts.



Report Distribution

Distribute reports to multiple formats including XLS, PDF, XML, and HTML without the need for coding.

Repository Explorer

Reduce report design time by reusing existing report objects across applications.

NEW Report Dependency Checker

Improve QA. Quickly find broken links, formula errors and dependency issues.

NEW Crystal Reports Server

Publish reports to the web for secure viewing, printing and exporting with a new report server option.

Kiss those long and tedious hours of integrating reporting into your applications goodbye. New Crystal Reports® XI, from Business Objects, adds a host of new functionality designed to reduce the time you spend creating, integrating and deploying reporting solutions. An enhanced designer, extended API and new deployment options, offer you and your end-users high quality viewing, printing and exporting with less effort.

NEW Free Runtime Licensing

Also new to Crystal Reports XI Developer Edition is a royalty free runtime license which allows for unlimited internal corporate deployment of the Crystal Reports .NET, Java™ and COM (RDC) report engine components without having to pay additional licensing fees for multiple servers or CPUs.

Get home sooner with Crystal Reports XI.

Visit www.businessobjects.com/dev/p26

for full product details, information on our new free runtime license or for a free eval download.

To contact us directly please call 1-888-333-6007.

```
if(n instanceof MonitorNotification) {  
    MonitorNotification mn= (MonitorNotification) n;  
    System.out.println("MonitorNotification  
        attributes:");  
  
    System.out.println("\t\t\t\t\tobserved mbean = "+  
        mn.getObservedObject());  
  
    System.out.println("\t\t\t\t\tobserved attribute = "+  
        mn.getObservedAttribute());  
  
    System.out.println("\t\t\t\t\ttrigger = "+  
        mn.getTrigger());  
}  
else {  
    System.out.println("Wrong notification type.");  
}  
}
```

Conclusion

This article describes an intelligent solution to integrate a custom multithreading framework using J2EE container architecture. Message-driven Beans implement the function of custom threads. The J2EE container manages the lifecycle of the MDB instances. By using a JMX notification model, we can monitor the runtime status of the MDB-based framework running in a distributed environment. ☸

Resources

- Tankut Bari Aktemur, Sinan Uakli, Hakan Onur. Work Thread Pool Framework. www.cs.bilkent.edu.tr/taosd03/Presentations/WTPFW.ppt
- Programming WebLogic Management Services with JMX: <http://e-docs.bea.com/wls/docs81/jmx/index.html>
- Paul Hyde, *Java Thread Programming*, Sams Publishing. (1999).
- Mark Artiges, Gurpreet Singh Bhasin, Bernard Cicone, Malcolm Garland, Saranathan Govindarajan, James Huang, Subramanian Kovilmadam, Kurnal Mittal, Paul J. Perrone, Tom Schwenk, Steven Steffen. *BEA WebLogic Server 8.1 Unleashed*, Sams Publishing (2003).
- Tim Francis, Eric Herness, Rob High, Jr., Jim Knutson, Kim Rochat, Chris Vignola. *Professional IBM WebSphere 5.0 Application Server*. Wiley (2003).

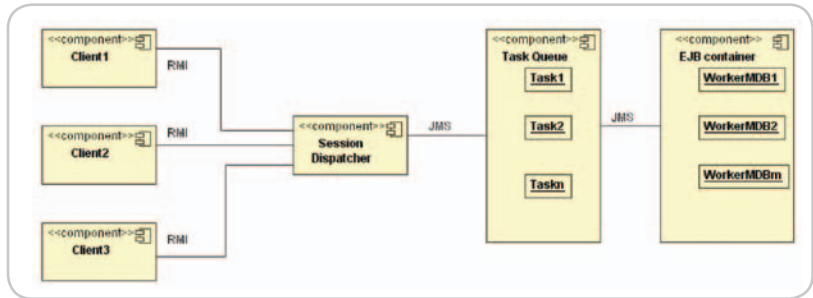


Figure 5 The conceptual view of the MDB-based CMF

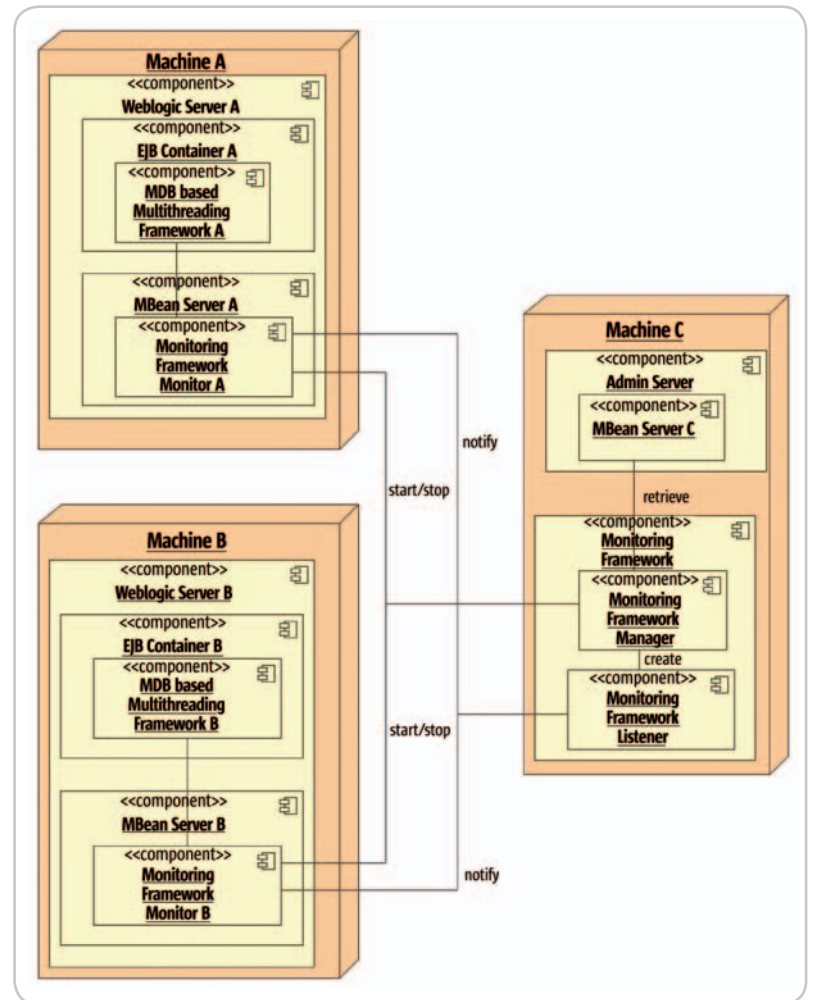


Figure 6 The architectural model of the JMX monitoring framework

Listing 1

```
public class TaskFIFOQueue{
    private static TaskFIFOQueue instance=null;
    private Vector queue;

    private TaskFIFOQueue(){
        queue=new Vector();
    }
    public static TaskFIFOQueue getInstance(){
        if(instance == null){
            instance=new TaskFIFOQueue ();
        }
        return instance;
    }
}
```

```
public synchronized void addOneTask(Task task){
    queue.add(task);
    notifyAll();
}

public synchronized Task getOneTask()
    throws InterruptedException{
    if(queue.isEmpty()) {
        wait();
    }
    Task task = (Task) queue.get(0);
    queue.remove(0);
    notifyAll();
    return task;
}
```

Get the complete picture



Discover the New

ILOG JViews Graphics Components

**First comprehensive Java graphics toolkit for Eclipse
Full Java Server Face (JSF) Support**

Build better GUIs in less time. Exceed your application requirements.

Reduce your development time & risk. Improve user experience & value.

Advanced BPM modeling and monitoring displays, data charts, gantt scheduling & resource displays, network & equipment displays, network diagrams, EMS/NMS telecom displays, custom dashboards and more. Whatever your display needs – for desktop or Web client – JViews has the solution.

Learn more. Test-drive an Eval. Call: 1-800-for-ILOG or go to:

- ILOG JViews Diagrammer <http://diagrammer.ilog.com>
- ILOG JViews Charts <http://charts.ilog.com>
- ILOG JViews Gantt <http://ganttt.ilog.com>
- ILOG JViews Maps <http://maps.ilog.com>
- ILOG JViews TGO <http://jtgo.ilog.com>



Changing the rules of business™

Remote Benchmarking with Servlets and JSF

by Anatoly Krivitsky

Wringing out performance

Powerful computers these days (including servers) are cheap compared to the “good old days.” In fact, they’re even cheap compared to what they cost a couple of years ago. Vendors are selling PCs whose CPUs clock above 3GHz for under \$1,400. Their memory and HDD capacity are also impressive. We’re basically looking at hardware that can **potentially** offer the performance of a mainframe for a fraction of a mainframe’s price. The operative word is **potentially**.

A gating factor is the set of software- and hardware-related components that sits between a CPU’s clock speed and the end user of the application. Consider the case of an application running on a server that provides results to an end user with Web pages. Here the set might be: the server (or servers), the server’s operating system, the application’s language, the application’s compiler, the tools that provide the variable content of Web pages (say, ASP.NET, Perl, servlets/JSP, etc.), the Web server, and the Internet provider’s additional soft and hardware. Mainframe-style performance on modern PCs and servers with powerful CPUs can be achieved only if the set allows it.

This article will introduce a simple new and powerful technology that will let you remotely choose the Web hosting provider that’s right for you. It will also show you how to use Whetstone for double type data (or **DWhetstone**) benchmarking together with JSF and servlets using the popular Tomcat 5 servlet container. Though there are a number of cool tools to measure performance (like JMeter from Apache for Java-related performance) the technology described here can be used with minimal changes across different languages and operating systems, giving

you a unique opportunity to measure different software and hardware combinations fairly and competitively.

Performance

Performance is either the factor or one of the factors that has to be taken into account when you choose all or some elements of the hardware/software set. It’s not enough just to measure the performance of parts of the set. Say we measure only the Web server’s performance. The results can be brilliant. The web server may be very good, but the other parts of the schema can be the bottleneck that prevents you from getting satisfactory results. We obviously will need to measure the performance of the whole set to get results from the point-of-view of the application.

What Are Benchmarks?

Benchmarks are useful tools that let you evaluate performance from the point-of-view of the end user. Benchmarks help you pick the optimal hardware and software. Benchmarks used to make a final decision should be the same for each language/compiler/tools for Web applications and Web servers. Obviously you need to run benchmarks with your own software/hardware combination rather than rely on third-party results.

Where to Find Information on Benchmarks?

A good place to start is <http://www.netlib.org/benchmark/>. This site also has references. It’s useful to do some research using search engines like www.yahoo.com and the keyword “benchmark.”

Whetstone

Whetstone is one of the oldest (and

therefore carefully debugged and tested) and most popular benchmarks. It is relatively simple and reliable. The specific form of this benchmark that we’re talking about here was described in my book *The New Technology of Making Scientific and Technical Documentation by IBM PCs*.

Remote Benchmarking

Remote benchmarking will help you to fill in gaps between measuring a server’s clock speed (which is usually provided by either the manufacturer or by the host provider) and the tools intended to measure the performance of your application (which is the next level of performance analysis).

Even before you start to measure the performance of your Web application using, say, JMeter, you need to pick the servers, server operating system, application language, application compiler and tools providing the variable content of the Web pages/Web server that fit your performance requirements.

You may need to resolve the following practical question: Suppose competing providers offered you the same rate and hosting on servers with the same clock speeds. You’ll need to select one of the following software combinations.

1. Apache and Tomcat 5 on a Linux box.
2. ONE and JRun on a Solaris box.
3. IIS and ASP.NET on an XP box.

How do you make the right choice? Well, remote benchmarking helps you answer this question. Some source code and further explanations on remote benchmarking can be found at my site <http://www.mycgiserver.com/~akrivitsky/Benchmarks.htm>. E-mail me at: akrivitsky@yahoo.com if you want the source code for this example or the source code of remote benchmarks using ASP.NET.



Anatoly Krivitsky has more than 24 years of experience in IT. He has written 20 published papers and books, holds five patents and has a PhD in computer science. Visit his Web site at www.mycgiserver.com/~akrivitsky.

E-mail: akrivitsky@yahoo.com

Web Applications Best Practice

To improve the quality of software it's useful to employ best practices – which, with Web application architecture, is MVC (model, view, controller). There are a number of MVC-based frameworks. These days, however, JSF from Sun Microsystems is gaining momentum and is supported by Apache, Oracle, IBM, Borland, and other prominent Java vendors. JSF is free.

Location, Location, Location...

Suppose you're going to develop a Web application on your Linux or Windows XP Pro boxes and install it on your provider's server. Here is what you need to do starting from the scratch.

First you need to download J2EE from Sun at <http://java.sun.com/j2ee/1.4/download.html> and install it.

In theory you can work with JSF at this point. See Sun's J2EE tutorial on how to do it. However, it's useful to use JSF with Tomcat. You may want to download the latest version of Tomcat from Apache and use the technique described below.

Tomcat binaries can be downloaded from <http://jakarta.apache.org/site/binindex.cgi>.

The latest Tomcat release is Tomcat 5. The differences between versions aren't important to this article. We assume you'll use a Tomcat 5 servlet container.

The installations described here are straightforward; no additional comments are needed.

From here on out for brevity's sake it's assumed you're using a Windows XP Pro box. What to do with a Linux box is pretty much the same and will also be described below.

To work with JSF and Tomcat 5 you need to download the JSF reference implementation JARs that are zipped in a file jsf-1_1.zip from Sun at <http://java.sun.com/j2ee/javaserverfaces/download.html>.

Unzip it in a directory where a set of subdirectories will be created. Suppose, for a moment, that you downloaded jsf-1_1.zip in C:\Documents and Settings\My Documents\jsf 1.1. A set of subdirectories in C:\Documents and Settings\My Documents\jsf 1.1\jsf-1_1 will be created when you unzip the file.

You will need the following files from C:\Documents and Settings\My Documents\jsf 1.1\jsf-1_1\lib for your Web application:

```
commons-beanutils.jar
commons-collections.jar
commons-digester.jar
commons-logging.jar
jsf-api.jar
jsf-impl.jar
jstl.jar
```

You will also need a standard.jar file that can be downloaded from <http://archive.apache.org/dist/jakarta/taglibs/standard/binaries/jakarta-taglibs-standard-1.0.1.zip>. You may want to use another location for more recent version. Unzip the downloaded .zip file in a directory and locate standard.jar. All these jars should be copied under {Tomcat Home}\webapps\{Directory with Your Application}\[auxiliary directory]\WEB-INF\lib. Usually {} brackets enclose mandatory values, while [] brackets enclose values that can be omitted. Tomcat Home is a full path to the directory that contains Tomcat. If Tomcat Home is C:\jakarta-tomcat-5.0.19, the directory with your application is DwhetSF, the auxiliary directory is WebContent, which will be written as:

```
C:\jakarta-tomcat-5.0.19\webapps\
DwhetSF\WebContent\WEB-INF\lib
```

Configuration

server.xml

Though it's possible to run JSF-based Web applications for remote benchmarking without changing Tomcat's server.xml, I strongly recommend that you change it when you're developing on your local box and ask your provider to change it on the server to make it more convenient to do other things (like working with databases using JNDI). So it's important to have experience related to working with server.xml.

The file is located under {Tomcat Home}\conf. For remote benchmark applications the only change you need to make in server.xml is to insert this line tag in the file:

```
<Context path="/{Directory with
Your Application}" reloadable="true"
docBase="{Tomcat Home}\webapps\{Directory
with Your Application}\[auxiliary direc-
tory]" debug="0" />
```

In our example above notation will be written as:

```
<Context path="/DwhetSF" reloadable="true"
docBase="C:\jakarta-tomcat-5.0.19\webapps\
DwhetSF\WebContent" debug="0" />.
```

This tag should be inserted after:

```
<Logger className="org.apache.catalina.logger.FileLogger" directory="logs"
prefix="localhost_log." suffix=".txt"
timestamp="true"/> tag and before </Host>
tag.
```

faces-config.xml and web.xml

These files should be located under {Tomcat Home}\webapps\{Directory with Your Application}\[auxiliary directory]\WEB-INF\.

The standard contents of web.xml can be used in most of your development. I recommend the following sample of web.xml: www.javaranich.com/newsletter/200404/files/web.xml.html

You will need to change faces-config.xml for each new application. For remote benchmarking, faces-config.xml can be written as shown in Listing 1. The listing will be commented in the next sections.

Property Files

These files aren't mandatory but can be useful for various messages. For brevity's sake, the property file related to the application is omitted.

JSFs

As you can see in Listing 1 we need input.jsp and response.jsp. The first page is for user input handling and validation; the second one is used for remote benchmarking output. The connections with the managed bean described below are provided by the JSF Expression Language expressions




Figure 1 Application's input



Figure 2 Application's results

that begin with the symbols #{. The JSPs are shown in Listings 2 and 3.

An auxiliary simple index.jsp is used to point the browser at input.jsp. Mentioned pointing is provided by the tag `<jsp:forward page="faces/pages/input.jsp" />`. The index.jsp is located under {Tomcat Home}\webapps\{Directory with Your Application}\[auxiliary directory\] (Listings 2–4 can be downloaded from www.sys-con.com/jdj/sourcec.cfm.)

Managed Bean

This bean provides the main computations for remote benchmarking. It gets user input from input.jsp and provides information to be used in

response.jsp. The source code is given in Listing 4.

You can use any IDE that supports J2EE 1.4 or even Ant (the Apache tool) and a command-line compiler to build the class from the source code. The built class should be put under {Tomcat Home}\webapps\{Directory with Your Application}\[auxiliary directory\]WEB-INF\classes\{package name}\

How to Test

To test you need to start Tomcat and point your browser at `http://localhost:8080/{Directory with Your Application}/`.

Figure 1 shows the input of the application and Figure 2 shows the results.

How to Deploy on a Remote Server

After you will test the application on your local box, you'll want to deploy it. Ask your provider if you can deploy .war files. If the answer is yes, create a .war file and ftp it in the appropriate directory on the remote server.

If you can't deploy .war files then you need to ftp class, jsps, the configuration, and, if needed, property files. Submit the changes in server.xml to your provider or do it yourself if you're allowed.

Conclusion

This article offers a simple but powerful JSF- and servlet-based Web application for remote benchmarking. You should now be able to do remote benchmarking and develop simple- to medium-level JSF- and servlet-based Web applications. I strongly recommend that you check the Web sites mentioned here for updates. ☺

References

- David Geary, Cay Horstmann. (2004). *Core JavaServer Faces* (Sun Microsystems Press Java Series). Pearson Education.
- Hans Bergsten. (2004) *JavaServer Faces*. O'Reilly.

Listing 1: faces-config.xml

```
<?xml version='1.0' encoding='UTF-8'?>

<!--
Copyright (c) 2004 Anatoly S. Krivitsky, PhD.
All rights reserved.
Conditional permission for free use of the
code.
As soon as above copyright notes are
mentioned,
the author grants a permission to any person
or organization
to use, distribute and publish this code for
free.
Questions and comments may be directed to the
author at
akrivitsky@yahoo.com and akrivitsky@usa.com.
Disclaimer of Liability
The user assumes all responsibility
and risk for the use of this code "as is".
There is no warranty of any kind associated
with the code.
Under no circumstances,
including negligence,
shall the author be liable
for any DIRECT, INDIRECT, INCIDENTAL, SPECIAL
or CONSEQUENTIAL
DAMAGES,
or LOST PROFITS that result from the use or
inability to use the
code.
Nor shall the author be liable for any such
damages including,
but not limited to, reliance by any person on
any
information obtained with the code
-->
```

```
<!DOCTYPE faces-config PUBLIC
"-//Sun Microsystems, Inc.//DTD JavaServer Faces Config 1.0//EN"
"http://java.sun.com/dtd/web-facesconfig_1_0.dtd">

<faces-config>

  <application>
    <locale-config>
      <default-locale>en</default-locale>
    </locale-config>
    <message-bundle>
      DwhetBeanMessages
    </message-bundle>
  </application>

  <managed-bean>
    <managed-bean-name>DwhetBean
  </managed-bean-name>
    <managed-bean-class>
      akrivitsky.DwhetBean
    </managed-bean-class>
    <managed-bean-scope>
      session
    </managed-bean-scope>
  </managed-bean>
  <navigation-rule>
    <from-view-id>
      /pages/input.jsp
    </from-view-id>
    <navigation-case>
      <from-outcome>success</from-outcome>
      <to-view-id>
        /pages/response.jsp
      </to-view-id>
    </navigation-case>
  </navigation-rule>
</faces-config>
```

SIAMESE FIGHTING FISH ATTACK AND KILL EACH OTHER IF PUT IN THE SAME ENVIRONMENT.

ENTERPRISE APPLICATIONS OFTEN BEHAVE SIMILARLY.

WINDOWS SERVER SYSTEM WITH .NET REDUCES THE HASSLE OF APPLICATION INTEGRATION.

CRM and supply chain applications really can live in harmony. How? With Windows Server System™ and .NET. The .NET Framework, an integral component of Windows Server System, is the development and execution environment that allows different components and applications to work together seamlessly. That means applications are easier to build, manage, deploy, and integrate. The .NET Framework uses industry standards such as XML and Web Services which

allow enterprise applications to be connected to infrastructure of any kind. In addition, the productivity-enhancing features of .NET, such as automatic mapping of data to and from XML, also help to simplify integration by reducing the amount of code required to get it all done.

Find out more about application integration with Windows Server System and .NET: Simply get the Connected Systems Resource Kit at microsoft.com/connectedsystems



WITH WINDOWS SERVER SYSTEM AND .NET, YOU CAN OVERCOME INTEGRATION ISSUES. HOWEVER, WINDOWS SERVER SYSTEM IS USELESS AGAINST THE SIAMESE FIGHTING FISH.

Microsoft
**Windows
Server System™**

JMS Web Services Meet the J2EE Connector Architecture

Reusing your back-end messaging infrastructure while enjoying the benefits of SOA

by Frances Zhao

This article discusses an implementation strategy that illustrates how organizations can use JMS and J2EE Connector technologies to reuse their infrastructure and lay a foundation that will let them reap the business and agility benefits of SOA.

For a Service Oriented Architecture (SOA) to be truly resilient, asynchrony must be built into its fabric. Most large organizations, however, have already invested in a back-end messaging infrastructure using products such as MQ Series, Tibco JMS, Oracle Advanced Queuing, Sonic JMS, or SwiftMQ. They don't want to replace that infrastructure to get to SOA; they want to reuse it. Enter J2EE 1.4 with two key technologies to enable Web Services and messaging infrastructure reuse: JAX-RPC, the built-in interoperability-tested J2EE API for building SOAP-based Web Services; and the J2EE Connector Architecture, now designed to support both inbound and outbound communications from Enterprise Information Systems.

Let's start with an overview of the related technologies, such as J2EE Connector Architecture (JCA), Java Messaging Service (JMS), Web Services, and Java API for XML Remote Procedure Calls (JAX-RPC). Then I'll present our implementation strategy and use a sample application to show how it works in detail.

J2EE Connector Architecture

JCA is a required J2EE 1.4 API. It defines a standard architecture for connecting the J2EE platform to heterogeneous Enterprise Information Systems (EISs) such as ERP systems, mainframe transaction processing systems, relational database systems, or JMS providers.

JCA defines system-level contracts that encapsulate important requirements for effective, scalable integration with EISs, such as connection pooling and transaction management.

The EIS side of these system-level contracts is implemented in a resource adapter, a system-level software driver that's used by an application server or a client to communicate and operate with an EIS. While a resource adapter is specific to the EIS it represents, it's not specific to a particular application server and can therefore be reused across any J2EE application server.

Java Messaging Service

JMS is a J2EE API for Java messaging clients. It provides two programming models: point-to-point and publish-subscribe. In the point-to-point model, one sender puts a message in a queue and it's delivered only to one receiver. The publish-subscribe model adds a broadcast mode in which any number of publishers can add messages to a topic, and any number of subscribers gets all the messages posted to the topics they subscribe to. JMS queues and topics are bound to a JNDI environment and made available to J2EE applications.

Web Services and JAX-RPC

Web Services are a set of Internet-standard messaging protocols, programming standards, and network registration and discovery facilities that expose business applications to other services and clients over the Internet in an interoperable fashion using XML messaging standards.

An important development in the J2EE community has been the standardization of the Web Services API in the J2EE platform with J2EE 1.4. The primary design intent of this standardization has been the natural adoption of the supported Web Services so that SOAs can take full advantage of the foundation provided by J2EE 1.4.

Web Services in J2EE 1.4 build on the underlying J2EE platform by adding the core standards SOAP and Web Service Description Language (WSDL) as first-class citizens. For J2EE developers, Web Services aren't a new concept but an effortless extension to the programming model they've been using for years.

The Web Services support in J2EE 1.4 encompasses the following major standards: JAX-RPC 1.1, SAAJ (SOAP Attachments API for Java) 1.2, Enterprise Web services for J2EE 1.1 (JSR-109), JAXP (Java API for XML Parsing) 1.2, and JAXR 1.0 (Java API for XML Registries). Of these, only JAXP was supported in J2EE 1.3.

With JAX-RPC, any Java class or stateless session EJB can be exposed as a Web Service. The specification defines how to use JAX-RPC to create and consume SOAP messages in Java, including issues such as serializing and de-serializing Java types with XML, dealing with SOAP attachments, and managing SOAP faults and headers.

JAX-RPC provides a client runtime that lets Java clients send and receive SOAP messages, and a server runtime for server-side Java implementations providing a Web Services API using

Frances Zhao is a product manager for the Oracle Containers for the J2EE team at Oracle. She focuses on Web services and enterprise application connectivity.

frances.zhao@oracle.com

SOAP. A JAX-RPC server implementation doesn't rely on a JAX-RPC client; rather, the client can be .NET, PL/SQL, or any other language capable of sending and receiving SOAP messages. Likewise, the JAX-RPC client can be used independently of a JAX-RPC server and can interoperate with other SOAP server-side implementations.

Implementation Strategy

Our implementation strategy involves three steps:

1. Configure a JMS resource adapter to work with your specific JMS provider or back-end.
2. Build a JMS application that accesses the JMS provider via the configured JMS resource adapter.
3. Publish your JMS application as a JAX-RPC-based Web Service and start enjoying the benefits of SOA.

Figure 1 illustrates a typical application architectural diagram following this implementation strategy. I will build the application from the right-bottom up in the diagram starting with Step 1. Then I'll move from right to left, bottom up, to complete the entire application, which will be accessible as a Web Service.

How It Works

Next, I will use an example application to show you in detail how this implementation strategy works. For demonstration purpose, I'll use the J2EE 1.4-compliant server Oracle Containers for J2EE (OC4J) and Oracle JDeveloper as the basis for the code and configuration file listings. I'll assume that the JMS provider is WebSphere MQSeries.

Application Scenario Example

Our example application consists of a session bean and a message-driven bean (MDB). The session bean provides an interface for sending a message (that contains a business service request) to an MQSeries queue that the MDB is listening to. When the MDB gets the message, it sends a reply message (that reflects updated business information or a simple acknowledgement) back to this queue, linking the received and reply messages using JMSCorrelationID.

The session bean's interface call is synchronously blocked until it verifies that all replies are correctly received. The synchronous call here is not a requirement - the MDB can also trigger a separate bean to process the reply messages. It's optional that the message receipt and reply be included in the same transaction.

Configuring the JMS Provider and Resource Adapter

First, I need to properly configure the MQSeries JMS provider and set up a JMS resource adapter to work with MQSeries. The resource adapter I used for this example is the OC4J JMS resource adapter, which is JCA 1.5-compliant. For real business applications, any JCA 1.5-compliant resource adapter that supports a desired JMS provider (e.g., MQSeries) would serve the purpose.

Configuring MQSeries mostly involves setting up the queues, topics, and connection factories using the JMSAdmin facility in MQSeries. We'll skip the details here. I also need to specify a resource provider (as opposed to an adapter) in OC4J for this application to use MQSeries.

The resource adapter configuration in OC4J involves three deployment descriptor files: `ra.xml`, `oc4j-ra.xml`, and `oc4j-connectors.xml`.

`ra.xml` is the standard J2EE deployment descriptor for resource adapters. For our application, I need to define the JMS queue and the queue connection factory's JNDI names. I also need to define a resource provider name that will be used as a prefix for JNDI lookup.

Listing 1 shows the configurations in `ra.xml` to enable access to the WebSphere MQSeries queue and connection factory. The resource provider name is set as 'MQSeries.' 'MQQ' and 'MQQCF,' which become the JNDI-bound JMS objects created using JMSAdmin of WebSphere MQSeries, can now be looked up by the application using an ordinary JNDI lookup. They can be found in the `.bindings` file created while configuring WebSphere MQSeries JMS Provider.

`oc4j-ra.xml` contains OC4J-specific deployment configurations for a resource adapter. For example, it contains EIS connection information as specified in the deployment descriptor of the resource adapter `ra.xml`, the JNDI name to be used, as well as connection pooling parameters and resource principal mappings (security-config element). Whenever you deploy a resource adapter, OC4J generates this file if it doesn't already exist in the archive. For our application, I only need to define a connection factory, and reference the JNDI location of the queue connection factory as in `ra.xml`.

Listing 2 shows the OC4J-specific configurations for queue connection factories in `oc4j-ra.xml`. (Listings 2-6 can be downloaded from www.sys-con.com/java/sourcec.cfm.)

`oc4j-connectors.xml` contains a list of resource adapters that have been deployed to OC4J. OC4J generates this file if it doesn't exist in the archive. For our application, I only need to define a queue, and reference the JNDI location of the queue as `ra.xml`.

Listing 3 shows the OC4J-specific configurations for queues in `oc4j-connectors.xml`.

Configure and Write the Application to Use the Resource Adapter

Once we've configured MQSeries and hooked it up with the JMS resource adapter, the next step is to configure and write the application that uses the resource adapter interfaces to send and receive messages from the MQSeries queues.

To enable EJBs (session bean and MDB) in this application to connect to the MQSeries queue, the OC4J-specific EJB deployment descriptor `orion-ejb-jar.xml` can be configured as in Listing 4 to allow the use of logical lookup names.

Note that the values of the location attributes here correspond to the values of the location attributes defined in `oc4j-ra.xml` and `oc4j-connectors.xml`. This ensures that the logical factory or queue names are mapped to those exposed via the JMS resource adapter.

The reference mapping names specified can now be used in the J2EE standard EJB deployment descriptor `ejb-jar.xml` as in

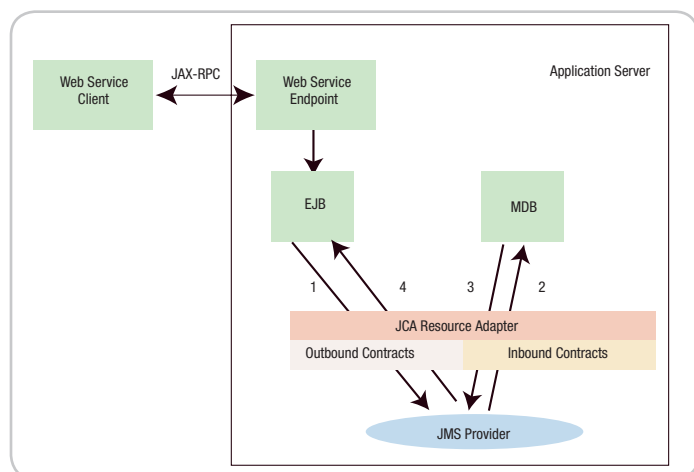


Figure 1 Sample application showing message flow from EJB through JMS

Listing 5. These are the names (factories or queues) that the session bean and MDB will use to access the MQSeries queues.

With the bean configurations done, I can write the code in the session bean and MDB to access the MQSeries queue via the JMS resource adapter, as shown in Listing 6.

Publish the Application as a JAX-RP-based Web Service

Now that you've created the EJB application accessing MQSeries via a JMS resource adapter, you're ready to publish it as a JAX-RPC Web service.

To publish the Web Service — that is, package and deploy it so external clients can send and receive SOAP messages to it — you must follow these steps:

1. Create the artifacts necessary for the server to understand that this set of Java classes is a Web Service. Conceptually, you're creating a binding between the inbound SOAP message, an invocation, and the returned results of a particular Java class and method. For stateless EJBs as in our example this is fully standardized in J2EE 1.4.
2. Create a WSDL document describing the service. This becomes the external contract that the Web Service conforms to. Most tools, regardless of programming language, can generate a Web Services client based on this document.
3. Create the deployment packaging so the JAX-RPC service can be deployed to a J2EE server. This step, often generically referred to as Java Specification Request (JSR) 109, describes the required artifacts every J2EE server needs to successfully deploy and execute a JAX-RPC Web Service.

Don't be intimidated by these steps. Many vendors offer GUI-based or command-line tools that automate them for you, for example, Oracle JDeveloper, BEA WebLogic Workshop, IBM WebSphere Studio Site Developer. Using these tools, the steps are as simple as identifying the class you want to publish,

selecting the methods you want to expose as Web Services, and defining the style of SOAP messaging (RPC/ Encoded, RPC/ Literal, or Document Literal). Since the procedures to follow are usually vendor-specific, I won't go into much detail in this article; suffice it to say that every J2EE 1.4-compliant vendor provides tooling for this purpose.

A final note: just as JAX-RPC standardizes the server side of J2EE Web Services, it provides a set of standard models for building JAX-RPC clients.

The goal of any Web Services client is to programmatically construct a SOAP message conforming to the specification laid out in the Web Services WSDL file. To that end, JAX-RPC implementations typically provide a WSDL-to-Java client generation tool, such as those found in both OC4J and Oracle JDeveloper.

Summary

In this article, I have presented an implementation strategy that illustrates how organizations can reuse their back-end messaging infrastructure (such as MQSeries, Tibco JMS, Oracle AQ, or Sonic JMS) and yet lay a foundation that will let them enjoy the benefits of SOA. Our strategy builds on two key technologies, JAX-RPC and the J2EE Connector Architecture, and consists of three steps: configuring a JMS resource adapter to work with your specific JMS provider or back-end; building a JMS application that accesses the JMS provider via the configured JMS resource adapter; and publishing your JMS application as a Web service. ☛

Resources

- J2EE Connector Architecture: <http://java.sun.com/j2ee/connector/index.html>
- JAX-RPC: <http://java.sun.com/xml/jaxrpc/>
- Java 2 Platform, Enterprise Edition (J2EE): <http://java.sun.com/j2ee>

Listing 1

```
<connector ...>
  <resourceadapter>
    <resourceadapter-class>
      oracle.j2ee.ra.jms.generic.JMSResourceAdapter
    </resourceadapter-class>
    ... ..

  <config-property>
    <config-property-name>resourceProviderName</config-property-
      name>
    <config-property-type>java.lang.String</config-property-type>
    <config-property-value>MQSeries</config-property-value>
  </config-property>
  ... ..

  <!-- Queue admin object -->
  <adminobject>
    <adminobject-interface>javax.jms.Queue</adminobject-interface>
    <adminobject-class>
      oracle.j2ee.ra.jms.generic.AdminObjectQueueImpl
    </adminobject-class>
    <config-property>
      <config-property-name>jndiName</config-property-name>
      <config-property-type>java.lang.String</config-property-
        type>
      <config-property-value>MQQ</config-property-value>
    </config-property>
    <config-property>
      <config-property-name>resourceProviderName</config-prop-
        erty-name>
      <config-property-type>java.lang.String</config-property-
        type>
      <config-property-value>MQSeries</config-property-value>

    </config-property>
    </adminobject>
  </connector>

  </config-property>
  </adminobject>

  ... ..
  <outbound-resourceadapter>
    <!-- non-XA Queue Connection Factory -->
    <connection-definition>
      <managedconnectionfactory-class>
        oracle.j2ee.ra.jms.generic.ManagedQueueConnectionFactoryImpl
      </managedconnectionfactory-class>
      <connectionfactory-interface>
        javax.jms.QueueConnectionFactory
      </connectionfactory-interface>
      <connectionfactory-impl-class>
        oracle.j2ee.ra.jms.generic.QueueConnectionFactoryWrapper
      </connectionfactory-impl-class>
      <connection-interface>javax.jms.Connection</connection-inter-
        face>
      <connection-impl-class>
        oracle.j2ee.ra.jms.generic.ConnectionWrapper
      </connection-impl-class>

      <config-property>
        <config-property-name>jndiLocation</config-property-name>
        <config-property-type>java.lang.String</config-property-
          type>
        <config-property-value>MQQCF</config-property-value>
      </config-property>
    </connection-definition>
  </outbound-resourceadapter>
  </resourceadapter>
</connector>
```



Pure **Java**
Pure **Excel**
Power for users
Control for IT
Pure **Paradise**

Formula One e.Spreadsheet Engine

*API-driven, embedded, 100% pure-Java,
scalable spreadsheet toolset*

Spreadsheets play an essential role in the business world. And now, they can also perform a vital function in your Java applications. How? With the Formula One e.Spreadsheet Engine, an API-driven, 100% pure Java toolset for embedding Excel spreadsheet functionality in your applications.

Excel-enable your Java applications

Use a state-of-the-art graphical development environment to build fully-formatted Excel report templates that include formulas, functions, colors, merged cells, even charts. It's fast, easy and effective.

Embed live, Excel-compatible data grids

Include interactive Excel forms and reports in your Java applications. Let users manipulate them using their spreadsheet skills and then commit the changes to databases or applications – or save them as an XLS file on their desktops.

Automate complex calculations and rules

Embed Excel spreadsheets that automate complex calculations and business rules on J2EE servers. Stop translating spreadsheet logic into Java code today, and start leveraging the development skills of your organization's spreadsheet experts.

Read and write server-based Excel spreadsheets

Give your users server-based spreadsheets populated with up-to-the-minute information directly from data-bases, XML files and enterprise applications. Get control of runaway data warehouses and spreadmarts now.

Make spreadsheets a part of your strategies

Visit us today at www.reportingengines.com to request a **free trial** of the e.Spreadsheet Engine. We'll show you how to make Excel spreadsheets a vital and productive part of your enterprise computing strategies.



www.reportingengines.com
sales@reportingengines.com
888-884-8665 + 1-913-851-2200

**FREE TRIALS,
DEMOS AND
SAMPLE CODE**

Three Sources of a Solid Object-Oriented Design

by Gene Shadrin

Design heuristics, scientifically proven OO design guidelines, and the world beyond the beginning

Object-oriented design is like an alloy consisting of a solid grounding in the object-oriented (OO) approach and implementing the best OO practices heavily laced with how to sidestep the OO pitfalls. The design process isn't only a matter of applying basic OO principles. One should go further and achieve a reasonable tradeoff between OO design principles and applicable design patterns. This article will discuss the relationship between three sources of a solid OO design and offer a starting point to understanding what is, in effect, a complex process. It will also serve as a review for experienced designers.

The OO Design Pyramid

When developers first turn to object-oriented development, they try to grasp the basics. It may take several project lifetimes. Getting a full sense of object orientation is a big shift in people's minds, especially for those who come from the procedural world. Such principles are like ABC books since they define the foundation of the OO world.

Once people get a sense of the basic principles, they start feeling the power of the OO approach. In our experience this is a dramatic moment for every OO designer and developer. The euphoria of possessing a powerful methodology can make them feel completely self-sufficient and they stop and don't try for perfection. It takes time to realize that there are significant issues beyond the basics that can be found in the accumulated wisdom of the OO developer community and are expressed in a set of design heuristics and scientifically proven OO design guidelines. Some OO designers jump to using design patterns everywhere like a cookie recipe. Although design patterns are important and useful

tools in building OO systems, they're just templates that define a common design language and significantly improve design quality, when properly used.

To build contemporary real-world enterprise-class systems, OO designers should (a) be proficient in basic OO principles, (b) master the principles of OO design, and (c) understand design patterns. Otherwise the chance of design flaws and total cost of system ownership increases. And, with growing adoption of Service Oriented Architecture (SOA), the robustness of the SOA services can depend directly on the solidity of the OO design of underlying components.

The pyramid in Figure 1 represents three sources of a solid OO design. It also defines a systematic view of design and uncovers dependencies between these sources.

Now let's discuss each level of this pyramid from an architectural and design point of view. Of course there's no way to explain these topics completely in such short article, but we'll try to highlight the necessity of paying attention to the dependencies and their impact on the quality of OO design.

Basic OO principles

The first level of the OO design pyramid is formed by a set of basic OO principles.

The most basic OO principles include encapsulation, inheritance, and polymorphism. Along with abstraction, association, aggregation, and composition, they form the foundation of the OO approach. These basic principles rest on a concept of objects that depicts real-world entities such as, say, books, customers, invoices, or birds. These classes can be considered templates for object instantiation, or object types. In other words, class specifies what an object can

do (behavior) and defines patterns for its data (state). Modeling the real world in terms of an object's state and behavior is the goal of the OO approach. In this case, state is represented by a set of object attributes, or data, and behavior is represented by the object's methods.

Now we'll discuss the application of basic OO principles to modeling.

Encapsulation encloses data and behavior in a programming module. Encapsulation is represented by the two close principles of information hiding and implementation hiding.

Information hiding restricts access to the object data using a clearly defined "interface" that hides the internal details of the object's structure. For each restricted class variable, this interface appears as a pair of "get" and "set" methods that define read-and-write access to the variable. *Implementation hiding* defines the access restrictions to the object methods also using a clearly defined interface that hides the internal details of object implementation and exposes only the methods that comprise object behavior. Both information and implementation hiding serve the main goal – assuring the highest level of decoupling between classes.

Inheritance is a relationship that defines one entity in terms of another. It designates the ability to create new classes (types) that contain all the methods and properties of another class plus additional methods and properties. Inheritance combines interface inheritance and implementation inheritance. In this case, *interface inheritance* describes a new interface in terms of one or more existing interfaces, while *implementation inheritance* defines a new implementation in terms of one or more existing implementations. Both interface inheritance and



Gene Shadrin is an enterprise architect with 17 years of software development experience, including object-oriented analysis, design, architecture, and development of real-life enterprise Java systems. He is a Sun Certified Enterprise Architect, PMP, and consults for government agencies and Fortune 500 organizations.

genes@eltrong.com

implementation inheritance are used to extend the behavior of a base entity.

Polymorphism is the ability of different objects to respond differently to the same message. Polymorphism lets a client make the same request of different objects and assume that the operation will be appropriate to each class of object. There are two kinds of polymorphism – *inheritance polymorphism*, which works on an inheritance chain, and *operational polymorphism*, which specifies similar operations for non-related out-of-inheritance classes or interfaces. Because inheritance polymorphism lets a subclass (subtype) override the operation that it inherits from its superclass (supertype), it creates a way to diversify the behavior of inherited objects in an inheritance chain, while keeping their parent-objects intact. Polymorphism is closely related to inheritance as well as to encapsulation.

Naturally, some OO principles are controversial in the sense that they contradict one another. For example, to be able to inherit from a class, one should know the internal structure of that class, while encapsulation's goal is exactly the opposite – it tries to hide as much of the class structure as possible. In real life the tradeoff between these two principles is a fine line that can't be established without stepping up to the next level in the design pyramid.

OO Design Principles and Heuristics

OO design principles and heuristics form the second level in OO design pyramid.

There are about a dozen OO design principles and four dozens OO design heuristics identified over the years by OO evangelists such as Grady Booch, Bertrand Meyer, Robert C. Martin, Barbara Liskov, and others. OO design principles define the most common scientifically derived approaches for building robust and flexible systems. These approaches proved to be the best tools in solving numerous OO design issues that can't be captured by fundamental OO principles.

The class structure and relationships group consists of the following design principles: the Single Responsibility Principle (SRP), the Open/Closed Principle (OCP), the Liskov Substitution Principle (LSP), the Dependency Inversion Principle (DIP), and the Interface Segregation Principle (ISP).

The *Single Responsibility Principle* specifies that class should have only one

OO Design Principles

Class structure and relationships

- Single Responsibility Principle (SRP)
- Open/Closed Principle (OCP)
- Liskov Substitution Principle (LSP)
- Dependency Inversion Principle (DIP)
- Interface Segregation Principle (ISP)

Package cohesion

- Reuse/Release Equivalency Principle (REP)
- Common Closure Principle (CCP)
- Common Reuse Principle (CRP)

Package coupling

- Acyclic Dependency Principle (ADP)
- Stable Dependency Principle (SDP)
- Stable Abstractions Principle (SAP)

reason to change. It's also known as the cohesion principle and dictates that class should have only one responsibility, i.e., it should avoid putting together responsibilities that change for different reasons.

The *Open/Closed Principle* dictates that software entities should be open to extension but closed to modification. Modules should be written so that they can be extended without being modified. In other words, developers should be able to change what the modules do without changing the modules' source code.

The *Liskov Substitution Principle* says that subclasses should be able to substitute for their base classes, meaning that clients that use references to base classes must be able to use the objects of derived classes without knowing them. This principle is essentially a generalization of a "design by contract" approach that specifies that a polymorphic method of a subclass can only replace its pre-condition by a weaker one and its post-condition by a stronger one.

The *Dependency Inversion Principle* says high-level modules shouldn't depend on low-level modules. In other words, abstractions shouldn't depend on details. Details should depend on abstractions.

The *Interface Segregation Principle* says that clients shouldn't depend on the methods they don't use. It means multiple client-specific interfaces are better than one general-purpose interface.

The package cohesion group consists of the following design principles: the Reuse/Release Equivalency Principle (REP), the Common Closure Principle (CCP), and the Common Reuse Principle (CRP). This group

deals with the principles that define packaging approaches based on class responsibilities (i.e., how strongly related the responsibilities of classes are).

REP makes release granularity equal to reuse granularity, *CCP* says classes that change together belong together, and *CRP* says classes that aren't reused jointly shouldn't be grouped together.

The package coupling group consists of the following design principles: the Acyclic Dependency Principle (ADP), the Stable Dependency Principle (SDP), and the Stable Abstractions Principle (SAP). This group deals with principles that define packaging approaches based on the packages' collaboration (i.e., how much one package relies on or is connected to another).

ADP prohibits forming cyclic dependencies among packages, *SDP* says package dependency should be allowed to reinforce package stability, and *SAP* says stable packages should be abstract packages.

Design heuristics derive from the practical experience of OO developers. They normally lie on top of design principles but can interrelate with them or even underlie design principles. Heuristics can extend design principles to several specific implementations. That's why they're on the pyramid along with design principles.

As design principles, heuristics are grouped by their application: class structure, object-oriented applications, relationships between classes and objects, inheritance relationships, association relationships, etc. Heuristics are less fundamental than design principles, but they clarify, explain, and expand design principles.

Both design principles and heuristics can be controversial. In other words, some design principles and heuristics have internal dissension, while others contradict each other. For example, conforming to the Open/Closed

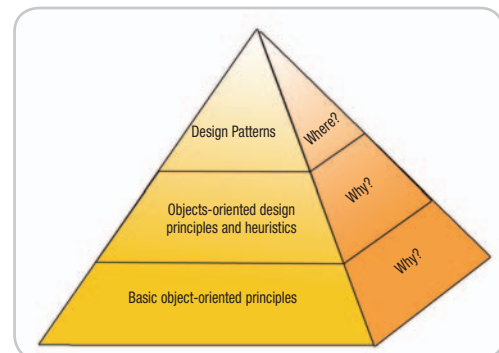


Figure 1 Design pyramid

Principle can be expensive and lead to unnecessary complexity – the class model should be pertinent to a specific context. What do designers get if they achieve conformance? The answer is the greatest benefits of the OO paradigm – flexibility, reusability, and maintainability. Another example – the Liskov Substitution Principle restricts the use of inheritance while the Open/Closed Principle embraces it.

But for all the controversy inherent in design principles and heuristics, they still give the OO designer such a powerful and systematic basis for robust design that the resulting OO model created with their “help” is of immeasurably better quality than the ones built on just basic OO principles.

Design Patterns

The top level of OO design pyramid represents design patterns.

There are 23 basic design patterns identified by “Gang of Four” (Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides), 21 core J2EE patterns identified by the Sun Java Center, 51 patterns of enterprise application

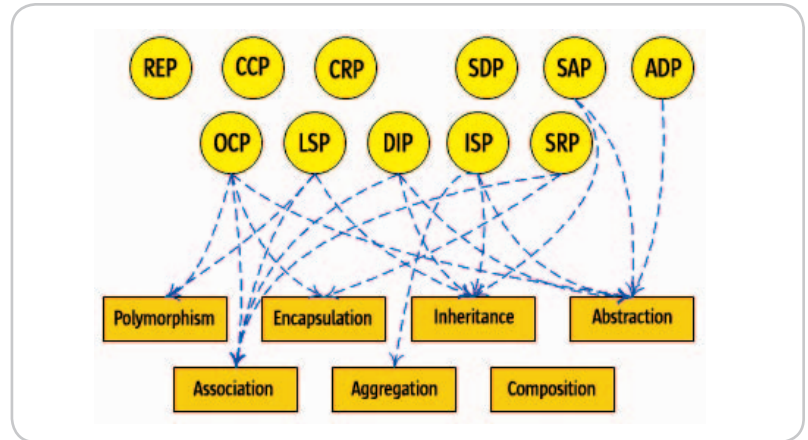


Figure 2 Dependencies between OO design principles and basic OO principles

architecture identified by Martin Fowler et al., and 65 enterprise integration patterns. There are also a lot of patterns specific to particular problem domains.

Design patterns represent common structured solutions to design problems solved in a particular context. They’re a guide to good design practices and span a wide range of solutions from general topics like object lifecycle and structure to more specific themes such as presentation, business, data, and integration tiers, data transfer, state management, message construction, routing, channels, and transformation. Each pattern describes intent, motivation, applicability, structure, participants, collaborations, consequences, and implementation, using one of the common notations (most often UML).

To manage design patterns better and simplify their application to real systems, all patterns are categorized. Categories reflect the approach to group them together. For example, the “Gang of Four” (GoF) patterns generally considered fundamental to all other patterns were categorized by their authors into three groups: Creational, Structural, and Behavioral. Java practitioner Steven John Metsker categorized these same patterns into different groups: Interfaces, Responsibility, Construction, Operations, and Extensions.

The rule of thumb is to try to apply patterns where application design would benefit from performance and flexibility. Sometimes, however, designers have to choose patterns based on just one “benefit.” For example, to improve performance, designers

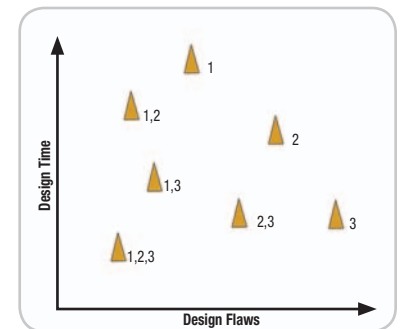


Figure 3 Design time vs quality (1 = basic OO principles, 2 = OO design principles, 3 = design patterns)

should always apply the core J2EE patterns of Data Access Object, Front Controller, Session Façade, Service Locator, Transfer Object, Transfer Object Assembler, Value List Handler, and Composite Entity.

In most cases, it’s possible to come up with appropriate design patterns for particular problems. Still, there are situations where design patterns either don’t exist or offer an inefficient solution. In that case, the solution may rest with design principles and heuristics.

Putting It All Together

The sections above described the content of each OO design pyramid level. Interdependencies between pyramid levels are represented by a complex graph that confirms their controversial semantics. As seen in Figure 2, which represents relationships between the first and the second levels (basic OO principles and OO design principles), some OO design principles depend on multiple basic OO principles while others are drawn based on other considerations.

The GoF patterns categorized by their authors:	GoF patterns categorized by Steven John Metsker:
Creational <ul style="list-style-type: none"> Abstract Factory Builder Factory Method Prototype Singleton Structural <ul style="list-style-type: none"> Adapter Bridge Composite Decorator Façade Flyweight Proxy Behavioral <ul style="list-style-type: none"> Chain of Responsibility Command Interpreter Iterator Mediator Memento Observer State Strategy Template Method Visitor 	Interfaces <ul style="list-style-type: none"> Adapter Façade Composite Bridge Responsibility <ul style="list-style-type: none"> Singleton Observer Mediator Proxy Chain of Responsibility Flyweight Construction <ul style="list-style-type: none"> Builder Factory Method Abstract Factory Prototype Memento Operations <ul style="list-style-type: none"> Template Method State Strategy Command Interpreter Extensions <ul style="list-style-type: none"> Decorator Iterator Visitor

Table 1 GoF patterns categories

Product	Company	Patterns		URL
Rational Software Architect	IBM-Rational	GoF	J2EE	www-306.ibm.com/software/awdtools/architect/swarchitect/index.html
Rational Rose XDE Developer for Java	IBM-Rational			www-306.ibm.com/software/awdtools/developer/java
Borland Together Architect	Borland	✓	✓	www.borland.com/together/architect/index.html
Borland Together Designer	Borland	✓	✓	www.borland.com/together/designer/index.html
Visual Paradigm for UML	Visual Paradigm			www.visual-paradigm.com/productinfovpumlee.php
MagicDraw UML	No Magic	✓*		www.magicdraw.com
Enterprise Architect	Sparx Systems			www.sparxsystems.com.au/ea.htm
System Architect	Popkin Software			www.popkin.com/products/system_architect.htm
Poseidon for UML	Gentleware			www.gentleware.com/products
Telelogic TAU/Architect	Telelogic			www.telelogic.com/products/tau/Architect/index.cfm
ArgoUML	(Open source)			argouml.tigris.org
* Partial support				

Table 2 Some OO design tools with Java code generation

As such, trying to draw the dependencies in such a graph is impossible. So a more appropriate methodology would be viewing these design pyramid levels and their relationships from the context of their application. Using this methodology, we can represent such relationships with this simile. If we take a car engine domain as an example, then we can think of the basic OO principles as the ones defining reciprocal-to-rotary motion conversion principles. This answers the “**why**” question and explains why the system works the way it works.

Presentation Tier

- Intercepting Filter
- Context Object
- Front Controller
- Application Controller
- View Helper
- Composite View
- Dispatcher View
- Service To Worker
- Business Tier
- Business Delegate
- Service Locator
- Session Façade
- Application Service
- Business Object
- Composite Entity
- Transfer Object
- TO Assembler
- Value List Handler
- Integration Tier
- Data Access Object
- Service Activator
- Domain Store
- Web Service Broker

Table 3 Core J2EE patterns

We can think of OO design principles and heuristics as answers to the “**what**” question because they unfold what to do to achieve design harmony. This is similar to the answer on what to do to build an internal-combustion engine.

At the top level of the OO design pyramid, we find the most effective approaches to solving generic and specific problems in certain contexts. We can think of this level as the one answering the “**where**” question. It’s like picking the most efficient engine constructions for particular consumer requirements.

So different levels of the OO design pyramid tackle different aspects of OO design. Experience shows that one can’t leave any of them out without running the risk of losing something important. Applying the elements from the upper levels without aligning them to the lower levels can lead to design flaws, while applying elements from the lower levels without knowing the upper levels can increase design time (potentially at a lower quality), since designers would be forced to “reinvent the wheel” (see Figure 3).

Conclusion

Because OO design elements focus on different aspects of OO design, coupled with their controversial and generic nature, knowing or even mastering only one or two levels of the OO design pyramid isn’t enough to develop contemporary, robust, flexible, extensible, and stable software. OO designers and developers have to take all levels of the pyramid into account and apply them in a systematic manner.

Applying OO design levels in a top-down approach (from design patterns to design principles and heuristics to basic OO principles) saves design time and increases design quality by ensuring that none of the important dependencies are lost. ☺

References

- *Thinking in Java, Third Edition* by Bruce Eckel, Prentice Hall PTR, 2002, ISBN: 0131002872
- *Agile Software Development, Principles, Patterns, and Practices* by Robert Cecil Martin, Prentice Hall, 2002, ISBN: 0135974445
- *Object-Oriented Design Heuristics* by Arthur J. Riel, Addison-Wesley Professional, 1996, ISBN: 020163385X
- *Design Patterns* by Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, Addison-Wesley Professional, 1995, ISBN: 0201633612
- *Design Patterns Java Workbook* by Steven John Metsker, Addison-Wesley Pub Co, 2002, ISBN: 0201743973
- *Core J2EE Patterns: Best Practices and Design Strategies, Second Edition* by Deepak Alur, Dan Malks, John Crupi, Prentice Hall PTR, 2003, ISBN: 0131422464
- *Patterns of Enterprise Application Architecture* by Martin Fowler, David Rice, Matthew Foemmel, Edward Hieatt, Robert Mee, Randy Stafford, Addison-Wesley Professional, 2002, ISBN: 0321127420
- http://www.objectsbydesign.com/tools/umltools_byCompany.html for a list of design tools.

Building Web Apps That Leverage Content Delivery Networks

by Alex Maclinovsky

Five different integration options

As the Web becomes an intrinsic part of the economy and our everyday lives, the success and survival of many businesses increasingly depend on the availability and accessibility of their core Web applications. Although a high degree of scalability and reliability can be achieved through the right combination of local and global redundancy, load balancing and sound application design, many companies turn to Content Delivery Networks or CDNs such as Akamai or Speedera. This article recounts experiences and lessons learned from developing an information portal that serves millions of users and leverages Akamai's CDN.

How CDN Works

Typically CDN providers augment the traditional Web infrastructure shown in Figure 1-a by introducing thousands of *edge servers* usually located at ISPs, carriers, backbones and other Web hubs around the world. They intercept the HTTP traffic directed to the sites of the network's customers and attempt to serve the requests from the closest possible location as shown in Figure 1-b. If the requested information can't be found in the cache, an edge server requests it from the origin site, passes it to the client, and then caches it to serve future requests.

It's important to know that page URLs are used as caching keys. If two pages have identical URLs, they would be considered the same page even if their content were different. Conversely, two identical pages with different URLs would be considered distinct and would be cached separately.

This basic service model is often complimented by premium services. In case of Akamai it includes *server mapping* and *cache hierarchies*. Its edge platform consists of more than 14,000 servers. Under high load conditions, the origin site could be swamped by the

requests from edge servers from different locations. Creating a dedicated *cache hierarchy*, which is done by establishing parent-child relationships among the edge servers, can mitigate this situation. In such hierarchies, children request content from their parents rather than the origin site. This provides caching on multiple levels, and can greatly reduce the number of edge servers that access the site itself. *Server mapping* involves dedicating specific servers in each data center to a particular origin site. This reduces the overall number of edge servers that access the site and improves the cache hit-rate of each server.

Today, many well-known sites are delivered from the edge by a CDN. An easy way to find out is to ping the same Web address through two separate providers, e.g. from home and the office. If it resolves to two different hosts outside the site's domain, they're likely to be edge servers. Another clue is when Netcraft reports an impossible platform combination such as this: <http://www.cdc.gov> was running **Microsoft-IIS** on **Linux** when last queried at 24-Sep-2004 10:01:22 GMT

CDN Integration Options

My first impression after reading white papers published by CDN providers and talking to their sales staff was that integrating a CDN into the solution architecture was easy and transparent. It looked like all that had to be done to make an application globally available was to sign the contract and provide the server and URL information. However, reality is rarely that simple, and CDNs, as most other optimizations, can be ineffective, even detrimental, if applied incorrectly.

Further analysis identified five possible levels of integration between Web applications and CDNs. They put different requirements on the Web application and have a major impact on end-user performance and the load on the application.

There are five possible levels of integration between a Web application and a CDN: *Asset Caching*, *Page Caching*, *Personalized Page Caching*, *Edge Side Includes*, and *Edge Computing*.

Asset Caching

Cacheable assets include file-based elements embedded in the pages and files that are downloadable from a site. They can be images, scripts, style sheets, applets, and static documents. This mechanism also includes serving the streaming media from the network's edge. It's completely transparent to applications and is the easiest integration method to use. Most CDNs are flexible enough to control caching objects by extension, file name, path, or domain, and let cached sites force the premature expiration of certain assets. The main drawback here is that, with the exception of streaming media, asset caching has little impact on either server load or the user's perception of performance. With this mode, each page is still served from the origin site and most load reduction occurs on the Web server, which is normally responsible for serving static assets. The application server still has to build every page, and the users have to wait for the entire roundtrip to complete before the page starts rendering in the browser.

Page Caching

The next level involves caching pages in their entirety. CDN usually allows the flexibility to control various aspects of caching and expiration down to the individual page level. Since edge servers cache pages by their URLs, for pages to be statically cacheable, they must render identical HTML when requested by different users or at different times within their *Time To Live* window. It's also important to limit the number of different URLs that represent each page. Having multiple URLs for the same page



Alex Maclinovsky is an enterprise architect specializing in J2EE solutions. For the last 15 years, he has focused on architecting and developing large distributed object systems on the enterprise, national and global scale. His professional interests include solution-oriented architectures, adaptive frameworks and OO methodologies. His professional profile and contact information can be found at www.geocities.com/maclinovsky/pro
maclinovsky@yahoo.com

can result in decreased efficiency and increased cost, since CDN providers usually charge by bandwidth. If applicable, Full-Page Caching is the most effective mechanism from all points of view. It offers the most significant load reduction on the entire Web application, dramatically reduces roundtrip times for cached pages, and ensures availability by serving the cached pages if the origin site goes down. The only drawback is that for consistent results, this method requires that an application maintain strict URL and content discipline.

Personalized Page Caching

Page Caching can be extended to *slightly* personalized pages (e.g., ones containing a user's name, or a link to 'My...' page) by using the <jesi:personalize> tag. This mechanism lets some user-specific values be calculated and inserted into a page at the edge. The information for such resolution comes from session cookies and other parameters such as browser locale from the request. It doesn't increase the load on the origin servers compared to Full-Page Caching, and imposes the same limitations on the application.

Edge Side Includes

Full Edge Side Includes (ESI) integration assembles volatile and highly personalized pages at the edge by caching static page fragments while delegating the generation of mutable content to the origin server. This mechanism provides the ultimate flexibility, but it requires extensive modifications to the Web site implementation, including the ability to render individual page fragments along with complete pages. These modifications don't map well to most Web architectures particularly MVC frameworks such as Struts. And, despite the claim that ESI was specifically designed with portals in mind, I have yet to find a successful implementation with any commercial portal platform, such as WebLogic and Vignette. To get the desired flexibility, ESI shifts a significant part of the load for generating every page to the origin server. If a page contains multiple includes, each one has to be requested separately, so page load times can actually be increased overall, especially when going over a highly latent connection.

Edge Computing

The most radical step in bringing dynamic content closer to the user is to generate it right at the edge. Akamai offers an *Edge Computing* solution based on WebSphere that deploys J2EE components such as JSPs and servlets to the edge of the network. Usually only the presentation components are pushed to the edge, however the offering includes Cloudscape – a lightweight database that can be used to store relatively static application data such as product catalogs. If used appropriately, this option is unbeatable for versatility, performance, and load management. But, building and managing massively parallel applications is a very complex task full of potential pitfalls and unmarked dangers. This option might also be unsuitable for non-technical reasons such as cost, network latency, and application security.

Building Page-Cacheable Applications

Given these considerations it becomes clear that, when applicable, *Page Caching* offers the optimal balance for improving performance, reducing the load on the target site, and preserving a Web application's original architecture. So from here on out we'll focus on building applications that work consistently

and reliably with the *Page Caching* mechanism of CDNs.

An Application's View of a CDN

When working through a CDN, applications still get HTTP requests from the Internet, but user demographics and behavior change dramatically. There's no longer a big and diverse population of "normal" users who surf pages sequentially and take time to read the content. Instead there's a small pack of "crazy" users, who look like they've bookmarked every page in the system and are jumping between them without any apparent logic or even time to digest the content. These users are cache servers, and their odd behavior comes from the *collective cache misses* of the actual users accessing the origin site through each server. Besides the virtual Attention Deficit Disorder, these new users also have multiple personalities – page requests that get through from a given edge server contain different cookies, browsers, and locales.

Avoiding Common Caching Caveats

As I said above, all caching is done strictly by URLs. To prevent erroneous or inconsistent behavior, Web apps should render identical HTMLs for all requests with the same URL. This

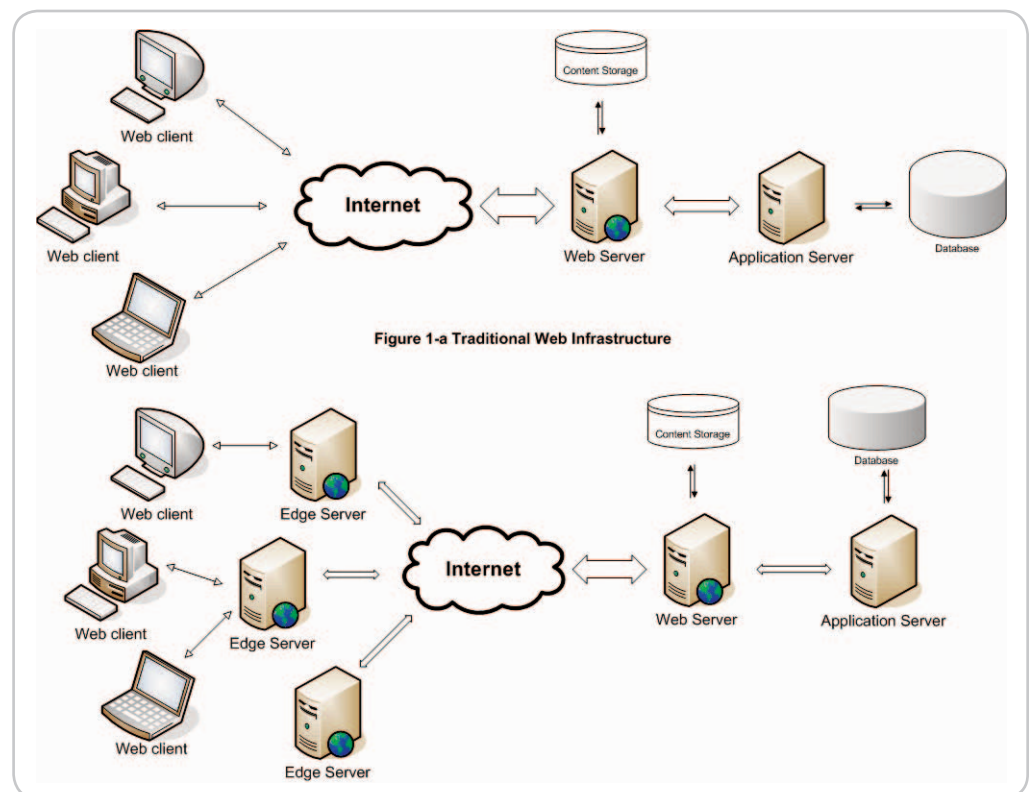


Figure 1 CSN's place in the Web infrastructure

includes rendering page for different users or at different moments of time. The best way to achieve this behavior is to **avoid using the session. Completely!** This dictum might sound strange and contrary to common wisdom and tradition, but given some thought, it'll make sense.

Session context is normally used to track user behavior and accumulate corresponding state. This information is then used to customize page content to reflect user history and preferences. By introducing a CDN into the application architecture, we no longer have the former and need to avoid the latter. All the data required to render a specific page should be kept in the page scope; everything that transcends it should be kept in the application or global scope. Every page that doesn't meet these requirements because of personalization or transactional behavior should be explicitly declared non-cacheable to the CDN. For consistency, it's best to combine all such pages in a separate part of the URL space, even in a separate domain that will be excluded from page-level caching altogether.

Remember Macavity the Cat whose main trick was *not being there*? It's challenging to show examples of **not** using `HttpSession` in a Java Web application. Especially since it's readily available from `HttpServletRequest`, and can be accessed from anywhere the request is

being passed. This includes servlets, JSP pages, custom tags, Struts actions, and other MVC components. Many of these are outside of the control of Java developers, but it's easy to verify that an application is *CDN-safe* using a Filter and `HttpServletRequestWrapper`.

Listing 1 details a simple implementation of `SessionCop` and `SessionRadar` classes that validate and enforce session non-use. They can be extended to allow stricter enforcement via exceptions, limit it to the cacheable region of the URL space, and support deactivation in a production environment.

This rule affects not only application design, but also user interaction. Some interface elements and behaviors that depend on accumulated state can no longer be supported. A good example is *breadcrumb* control.

Breadcrumbs are often used in portals to show users their location in a taxonomy. If the taxonomy is multi-hierarchical, i.e., navigation nodes can have more than one parent, there are two common ways to build breadcrumbs. It can be done statically by defining a *primary* parent for each node or dynamically based on the actual *click path* that the user took to get to the page. Naturally, the latter isn't compatible with *Page Caching*, and your user experience team should be cautioned to avoid creating unrealistic user expectations.

The other consequence of URL-based caching is that the pages whose URLs differ are considered distinct and are cached separately even if their content is identical. This includes even minor differences in parameter values. For example, if a Web server is configured to add the session ID as a URL parameter, there will be no cross-user caching and the edge servers will be swamped because they'll be caching endless copies of the same pages. Even the order of parameters is important. So caching effectiveness can be significantly improved by establishing and maintaining a strict parameter order.

The best way to achieve this is by using a single URL generator across the entire application. The JDK package `java.net` has a number of classes for URL manipulation, in particular the one appropriately named `URL`. This class includes methods for parsing URL strings and assembling URLs from individual components. It's useful for simple tasks like adding an application root or reference to a URL string. Unfortunately, it falls short on query parameter manipulation, let alone ensuring their consistent ordering. `URL` is declared final as are most primitive JDK classes, which prevents us from extending its functionality.

As a result we had to design our own Smart URL utility as shown in Figure 2. Class `SmartURL` is similar to the one in the JDK; however the attribute query is no longer a simple string, but an instance of `URLQuery`, which includes a list of `URLParams`. This design leverages the *strategy* pattern. The `URLStrategy` class determines all URL-generation policies, including sequence number versus the alphabetical ordering of parameters and the use of the parameter priority field so certain parameters such as the Struts action name always comes first. It's used by `SmartURL` and `URLQuery` when rendering the final URL string. We found `SmartURL` quite useful in most J2EE Web apps, even if they weren't delivered through a CDN.

Another subtle pitfall can cause unexpected session expiration. Consider a site that combines cacheable information pages with non-cacheable personalized or transaction ones. If security is important, user credentials are usually kept in a session configured to expire after a certain period of inactivity. Let's assume the site is configured to expire after 15 minutes of inactivity, and a user starts from non-cacheable *my* entry page. She's challenged,

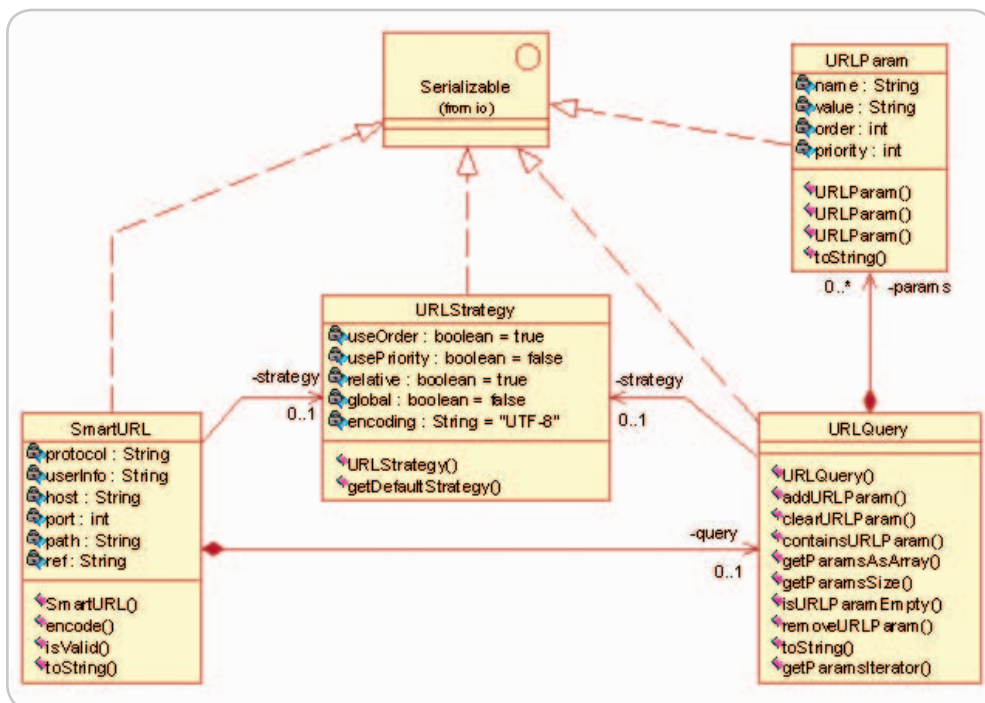


Figure 2 Smart URL Utility



Think Crystal is a good fit for your Java application? Think again.

Think JReport.

Forcing a Windows reporting solution into a J2EE environment is never going to result in a perfect fit. Only JReport is built from the ground up to leverage J2EE standards and modular components for seamless embedding in any Web application.

JReport is ready out-of-the-box, with all the tools necessary to empower your application with any reporting requirement. From reusable, shared report components to flexible APIs, JReport is the most complete embedded reporting solution available.

With the ability to scale to multi-CPU and server clusters, JReport is a perfect fit for any reporting workload. Load balancing and failover protection provide peak performance and uninterrupted access to critical business data. In addition, JReport integrates with any external security scheme for single sign-on.

JReport's ad hoc reporting lets users access and analyze data on demand, from any browser. And, with cascading parameters, embedded web controls for dynamic sorting and filtering, drill-down and pivot, JReport ensures users get the information they want, when they want it, and how they want it.

See for yourself why over half of the world's largest organizations have turned to JReport to enable their J2EE applications with actionable reporting.

When it comes to embedded Java reporting, JReport is the perfect fit. Download a FREE copy of JReport today at www.jinfonet.com/jd5.



logs in, and then proceeds to browse through the information pages that are cached and served from the edge server. After 20 minutes of browsing like this, if she visits a personalized page, she'd be challenged again. This counter-intuitive behavior happens because while the user was surfing cached pages, none of her requests reached the origin server, and her session expired due to inactivity. There are two possible workarounds for this issue.

The first one requires using If Modified – Send (IMS) queries in conjunction with Page Caching. If the *Time To Live* for all pages is set to zero, they'd still be cached. However, before sending each one to the user, the edge server will send an IMS query back to the origin site. These queries convey HTTP request information and maintain time-sensitive sessions while users are browsing cached pages. But this comes at the price of significantly increasing the load on the origin servers. The site should be able to process IMS messages and determine when to actually expire pages. This approach will diminish user performance, even in the case of a cache hit, because the edge servers will have to wait for the origin server to reply to each IMS query.

To handle IMS queries correctly in Java, we need to override the method `getLastModified(HttpServletRequest request)` from `HttpServlet`. The servlet container uses this method to determine when the servlet content was last modified. The default implementation simply returns the current time, which means the content will be considered expired and result in regenerating each page every time. Listing 2 contains a simple implementation of the `CDNServlet` class that provides a fixed time-to-live window for all content generated by the derived servlets. If you have an easy way to determine when specific content expires, this implementation can be extended by overriding the `getDelay(HttpServletRequest request)` method. When using an MVC framework such as Struts, the framework's *front controller* servlet should be extended the same way. Please note that this approach won't work for JSPs, which are navigable directly, as in MVC Model I, but will work fine with JSP forwards. (Listing 2 can be downloaded from www.sys-con.com/java/sourcecc.cfm.)

There's a simple non-technical alternative that doesn't impose any limita-

tions and is often sufficient. If cached and non-cached pages have different a look-and-feel and reside in separate domains, e.g., www.domain.com versus my.domain.com, most users won't have the **expectation** that a single session is maintained across the two.

Conclusion

When building a Web application that will be served through a Content Delivery Network, it's important to determine exactly what will be cached and delivered from the edge, and design the application accordingly. *Asset Caching* is least effective but universally applicable, and can be used without any limitations. *Page Caching* and *Personalized Page Caching* are very effective, but to work reliably and consistently they require cached applications to adhere to the basic rules outlined in this article. Maintaining strict URL discipline isn't necessarily a limitation but a good design principle. It will make

applications not only cacheable, but also easily bookmarkable (by users) and indexable by search engines. *Edge Side Includes* and *Edge Computing* are very powerful caching solutions for highly personalized and transactional applications, but they impose radical changes on the architecture and are incompatible with many Web technologies and application platforms. ☛

References

- Information about specific Content Delivery Networks can be found on the vendor sites: <http://www.akamai.com/en/html/technology/overview.html> and <http://www.speedera.com/primary/Tech/Over.htm>.
- <http://uptime.netcraft.com/up/graph> offers useful tools for probing sites and determining their hosting and platform information.
- Additional information on ESI can be found on <http://www.esi.org/>.

Listing 1

```
public class SessionRadar
    extends HttpServletRequestWrapper
{
    public static final String MSG =
        "Illegal use of session on cacheable page";

    public SessionRadar(HttpServletRequest request)
    {
        super(request);
    }

    public HttpSession getSession()
    {
        logger.error(MSG);
        // add stricter enforcement here
        return null;
    }

    public HttpSession getSession(boolean create)
    {
        return getSession();
    }
} // class SessionRadar

public class SessionCop implements Filter
{
    public void init(FilterConfig filterConfig)
        throws ServletException {}

    public void doFilter(ServletRequest request,
        ServletResponse response,
        FilterChain chain)
        throws IOException, ServletException
    {
        // add logic to deactivate in production
        // add logic to limit enforcement to
        // cacheable URLs
        ServletRequest wr=new SessionRadar(request);
        chain.doFilter(wr, response);
    }

    public void destroy() {}
} // class SessionCop
```

The Last Time You Saw Something This Incredible, It Was Science Fiction

FREE
(Limited Time)
NitroX JSP Editor
for Eclipse
Download at:
www.m7.com/jspfree

NitroX™ Web Application Development for Eclipse

NitroX's AppXRay™ penetrates all layers of your web application and helps annihilate your web application development problems!

NitroX AppXRay unique features include:

- Debug JSP tags, Java scriptlets, jsp: include, etc. directly within the JSP
- Advanced JSP 2.0 and JSTL support
- Advanced JSP editor – simultaneous 2-way source and visual editing with contextual code completion
- Real time consistency checking across all layers (JSP, Struts, and Java)
- Advanced Struts support – source and visual editors for Validation Framework, Tiles and Struts configuration
- AppXnavigator™ – extends Eclipse hyperlink style navigation to JSP and Struts
- AppXaminer™- analyze complex relationships between ALL web artifacts
- Immediate access to variables at all levels of the web application

Download a free, fully functional trial copy at: www.m7.com/d7.do

Copyright © 2004 M7 Corporation. All rights reserved. All M7 product names are trademarks or registered trademarks of M7 Corporation. Java and all Java based marks are trademarks or registered trademarks of Sun Microsystems. IBM, WSAD, WSSD are trademarks or registered trademarks of IBM.



Coding Business Logic in Excel

by Jerason Banes

Morphing an Excel spreadsheet into Java business logic

How many times have you coded a financial, engineering, or pricing calculator and used an Excel spreadsheet as a reference? What if you could take that Excel spreadsheet and make that the business logic for a Java application? The e.SpreadSheet API from ReportingEngines makes that dream a reality.

Formula One e.SpreadSheet Engine is a complete API for reading and saving spreadsheets. It can load a complete spreadsheet into memory, and do data changes, calculations, even charting, using nothing more than the formulas coded into the spreadsheet!

Even more impressive is its ability to create spreadsheets from scratch. With this functionality, reports can be delivered the same way they were received – in a spreadsheet.

In this article, I'll take you through a simple example of using a spreadsheet for common business logic. I'll use the JSP Tag Library APIs to make the code easy to follow along. But don't be fooled. The full API lets you do these calculations in EJBs, Swing Applications, or any other code that needs spreadsheet functionality.

Requirements

To use the code in this article, you'll need to download and install the e.SpreadSheet engine from <http://www.reportingengines.com>, and you'll need an application server capable of running JSP pages. If you don't have one readily available, you can download Apache Tomcat from <http://jakarta.apache.org>.

You'll also need a copy of Excel or OpenOffice to create the documents. Alternatively, you can download ReportingEngines' companion product e.SpreadSheet Designer. Designer is a GUI-based on the e.SpreadSheet engine. Its features are like Excel's and it lets you graphically design a spreadsheet from a database, XML source, or text file.

After you install the necessary

software, you'll need to extract the fltaginstall.war file and deploy it in Tomcat. This file contains everything a Web app needs to use the e.SpreadSheet APIs, so we'll use it as a shortcut for learning. You can find the file in the install directory under "eSS11/jars."

Creating the Spreadsheet

In this example, we're going to leverage a spreadsheet that does a useful – but not widely known – calculation called a "weighted average." The idea behind a weighted average is that a second number (such as the quantity of items) is used to give more or less importance to the numbers being averaged.

This function is commonly used to understand the true average price per product sold. For example, a grocery store may sell thousands of packs of bubblegum a week, but less than a hundred boxes of crackers. Using a normal average, the price of the crackers would raise the "average price per product," even though it sells nowhere near as well as the gum. With a weighted average, the number of items sold is properly figured into the average so that the store is aware of its real profit margins.

A weighted average can be easily calculated in Excel by using the formula, "SUMPRODUCT(RANGE1,RANGE2)/SUM(RANGE2)"; where RANGE2 is the quantity or "weighting" of the numbers. Let's create a spreadsheet to test this.

Open Excel or one of the other spreadsheet programs mentioned above. Place the following titles in row 1 of columns 'A,' 'B,' and 'C':

"Item," "Price," and "Units."

We'll use the first column to identify the product, the second for the product's price, and the third for the number of units sold. Feel free to enter data in the three rows. When you're done, the rows should look something like those in Figure 1.

Now, let's create a cell to display a regular average and a weighted average. Click on cell "D2" and type "=AVERAGE(B2:B5000)" in the calculator bar. If you used the data in Figure 1, "D2" should now show an average of \$14.12. Now click on cell "D3" and type "=SUMPRODUCT(B2:B5000,C2:C5000)/SUM(C2:C5000)." Again, if you used the data in Figure 1, you should have a weighted average of \$8.28. That's quite a difference!

Save the spreadsheet in the web application folder with the name "average.xls", and we will look at loading it into a JSP page using the e.SpreadSheet API.

Displaying the Data

With our spreadsheet safely saved as part of our Web app, we can create a JSP file to display its contents. Create a new JSP file in the Web app directory, and call it "average.jsp." Open it in your favorite text editor or IDE.

Put the following line of code at the top of your JSP file:

```
<% taglib uri="fltaglib" prefix="fl"%>
```

This line references the e.SpreadSheet tag library, allowing our JSP file to access its broad array of functions.

Next, we'll load the Excel file from disk with the following tag:

```
<fl:LoadBook book="Average" source="average.xls"/>
```

This tag tells the e.SpreadSheet engine we want to load the "average.xls" file, and that we want to let other tags reference it by the name "Average." This tag will keep the spreadsheet in memory

	A	B	C
1	Item	Price	Units
2	Pen	\$2.12	150
3	Gear	\$10.25	50
4	Clock	\$15.00	72
5	Dishes	\$20.22	10
6	Radio	\$23.00	15

Figure 1



Jerason Banes is a long-time Java developer and architect who enjoys Java architectural challenges. He is currently working on a cross-platform, cross-vendor database interface product known as DataDino Database Explorer.

jbanes@gmail.com

until the session expires or the “UnloadBook” tag is called. By default, the engine will ignore the “LoadBook” tag if a spreadsheet by the same name has already been loaded, so there’s no need to worry about losing any modifications you’ve made. This behavior can be overridden with the “reload” attribute.

Now that the prep work is done, we can finally display the data. We can use the “InsertRange” tag to populate an HTML table with data from the spreadsheet automatically. All we need to explain to the tag is the range and formatting. Try adding the following tag to the JSP page:

```
<f1:InsertRange book="Average" range="A1:C6"
table="true" spreadsheetformat="true"/>
```

That’s all there is to it! If you save your page and access it in a Web browser, you should see a table with the data from the “average.xls” spreadsheet. Basically, we’ve told the spreadsheet engine to pull cells A1 through C6 from the spreadsheet named “Average” and display it in a table. The “spreadsheetformat” attribute was also used to retain the cell formatting from the original spreadsheet. But what about the averages?

As one might expect, the “InsertRange” tag has a cousin called “InsertCell”. Instead of generating the complete table with formatting, “InsertCell” merely pulls the data for a single cell and plops it into the sheet. The following code will display the average and weighted average for the data:

```
Average Price:
<f1:InsertCell book="Average" cell="D2"/>
<br>
Weighted Average Price:
<f1:InsertCell book="Average" cell="D3"/>
```

If you save the JSP file and refresh your web browser, you should see both the average and weighted average of the data.

In and of itself, that may not seem all that odd, but something extraordinary just

occurred. When the engine displayed the value of the two cells, it displayed the results of their calculations and not the formulas themselves! This becomes more interesting later on when we begin to add or subtract from the spreadsheet.

In cases where you actually want the original formula, it can be extracted from the spreadsheet using the “GetCell” tag. In fact, a great deal of information can be extracted by using the “Get” and “ForEach” tags. The “Insert” tags are primarily for the convenience of outputting the formatted results.

Modifying the Spreadsheet

We’ve seen that we can read data out of the spreadsheet. Next we’ll cover modifying cells in the spreadsheet. The ability to modify spreadsheets is where the engine shines. It lets you plug in arbitrary values then retrieve the resulting calculations.

The key to modifying a document is to use the “SetCell” tag. For example, if we wanted to add a row of data to our spreadsheet, we might write the following code:

```
<f1:SetCell book="Average" cell="A7"
entry="Cola"/>
<f1:SetCell book="Average" cell="B7"
entry="2.25"/>
<f1:SetCell book="Average" cell="C7" entry="129"/>
```

This will create a new entry for cola sales, with 129 units sold at \$2.25 apiece. If we then print out the weighted average, we’d see that the value has changed significantly.

To let you experiment with this, I’ve made some simple modifications to our existing JSP page. You can see the complete code for this in Listing 1. The modifications merely provide a set of text-entry fields for adding rows to the spreadsheet. When the page is submitted, the fields are used to populate a new row, and the averages are updated automatically.

Charting the Numbers

One of the most impressive features in the

e.SpreadSheet engine is its ability to extract, display, and update charts embedded in Excel spreadsheets. The catch is you’ll need to install the e.SpreadSheet Designer. While the full API lets you loop through each chart, the JSP tag library requires you to name the charts for identification. Since this isn’t a standard feature in Excel, you’ll need to use the Designer to apply the name to the chart.

An easy way to demonstrate e.Spread-Sheet Engine’s charting features is to create a bar graph that compares the normal average to the weighted average. Open the average.xls file in your spreadsheet program and highlight cells D2 and D3. Next, select “Chart...” from the “Insert” menu and click on “Finish.” You should see a bar chart in your spreadsheet.

Save the spreadsheet and open it in the e.SpreadSheet Designer. Click on the chart to highlight it then click on the “Format” menu-bar item. Select “Object” from the dropdown and select the “Properties” tab. In the box marked “name” enter “AverageChart” and click on OK. Save the spreadsheet again, and you’re ready to use it in the JSP page.

To use this version of the spreadsheet, you’ll only need to make one change to the source in Code Listing 1. Add the following line of code to the end of the JSP file:

```
<f1:InsertChart book="Average"
chart="AverageChart"/>
```

This code uses the “AverageChart” name to find our chart and automatically insert it into the HTML output. When you run the JSP page, note that the chart automatically updates every time you add an item to the list.

Final Thoughts

As you can see, the e.SpreadSheet Engine has a rich set of APIs and we’ve barely scratched the surface. If you find yourself writing any sort of mathematical application or are in need of a good charting solution, e.SpreadSheet Engine may be the perfect solution. ☺

Listing 1

```
1<%@ taglib uri="fltaglib" prefix="f1"%>
2
3<f1:LoadBook book="Average" source="average.xls"/>
4
5<br>
6int rows = 6;
7String[] cells = {"A", "B", "C"};
8String[] values = {"columna", "columnb", "columnc"};
9
10if(request.getParameter("rows") != null)
11{
12    rows = Integer.parseInt(request.getParameter("rows")) + 1;
13    for(int i=0; i<cells.length; i++)
14    {
15        cells[i] = cells[i] + rows;
16        values[i] = request.getParameter(values[i]);
17    }
18    <f1:SetCell book="Average" cell="<%= cells[i]%>"
19        entry="<%= values[i]%>"/>
20 }
21 }
22 %>
```

```
23
24<form>
25<table border="0" padding="2" cellspacing="0">
26    <% String range = "A1:C"+rows; %>
27    <f1:InsertRange book="Average" range="<%= range%>"
28        table="false" spreadsheetformat="true"/>
29    <tr>
30        <td><input name="columna" size="12"/></td>
31        <td><input name="columnb" size="12"/></td>
32        <td><input name="columnc" size="12"/></td>
33    </tr>
34</table>
35<br>
36<input type="hidden" name="rows" value="<%= rows%>"/>
37<input type="submit" name="Add Row"/>
38
39Average Price:
40<f1:InsertCell book="Average" cell="D2"/>
41<br>
42Weighted Average Price:
43<f1:InsertCell book="Average" cell="D3"/>
44<br>
```

AOP-Enabled ESB

Updating the enterprise nervous system

by John Gilbert

Service Oriented Architecture (SOA) is not so much a new technology as a new state of mind. Technology for implementing business logic in the middle tier and exposing it as a service has been around for years. Yes, the technology is more standardized now and more widely accepted. What's really new is the growing practice of approaching SOA with an enterprise-wide mindset in which an organization takes a holistic approach to identifying the services they provide.

The rise of the Enterprise Service Bus (ESB) is linked to this new enterprise-wide view. ESB and SOA are complementary. One of the tenets of SOA is loosely coupling services. Message-Oriented Middleware (MOM) is used to connect decoupled systems. An ESB is created by using standards-based MOM providers consistently across an enterprise to glue decoupled services together. When used together this ESB-powered SOA enables the real-time enterprise.

The exact nature of an ESB is widely debated. Some people see it as a product, others as just an architectural design pattern. But most agree that an ESB can be thought of as the nervous system of the enterprise. It's the enterprise state machine that gets its stimulus in the form of business events as actors invoke services or exceptional conditions arise. These events flow across the bus and are evaluated based on business rules. The rules glue the decoupled services together to create a business process. As the rules invoke additional services more events are triggered and the whole cycle begins again.

The simplest way to explain the execution of a state machine is seen in Figure 1. An actor invokes an action that then fires an event. The event is evaluated against the state of the system based on the conditions or rules of the system. Conditions that evaluate to true then invoke another action and so on.

Regardless of whether an ESB is a product or not it's imperative that services are designed so they can plug easily into the ESB ensuring that all services publish an event to the ESB when they're invoked. This kind of crosscutting concern is best done using Aspect Oriented Programming (AOP), which dynamically inserts new code into existing code. As a result legacy code can be enhanced and new code can be developed more cleanly.

This AOP-enabled ESB can be explained best by an example. There are several AOP frameworks available. Which one you choose depends on how well it's already integrated with the tools used in your environment. For this article I used the AOP features included in the new EJB 3.0 specification and the JBoss preview implementation of the spec. I implemented several services using EJB 3.0 stateless session beans. The services are decorated with an AOP Interceptor. This interceptor gathers the method's invocation data and publishes it as an event to the ESB. The service bus is implemented

using a JMS Topic and a Message Driven Bean (MDB) that subscribes to the topic and passes the event to a JSR-94 compliant Rules Engine. Example rules are provided that further invoke additional EJB services. The complete code example can be downloaded from www.sys-con.com/java/sourceec.cfm.

Example

For the example, let's say BT is a supplier of Bluetooth gadgets. It has modeled its business process (Figure 2) and its business entities (Figure 3) and want to implement them with an ESB powered SOA.

In their business process customers submit orders via a portal. Once received, an order is provisioned, the customer is notified, and the shipping department is told to ship the order. Shipping is a manual task. This means the process won't move forward until an employee from the shipping department marks the task complete. Then the customer is notified and the order and the product quantities are updated.

What AOP Entails

Aspect Oriented Programming (AOP) is the name given to the approach of dynamically inserting new code into existing code. Typical uses of AOP are referred to as *crosscutting concerns*. These are secondary requirements that need to be applied consistently across all code. Classic examples are logging and security. Other examples can include persistence, validation, caching, and, in this article, publishing an event to a JMS topic.

The actual code that's inserted is referred to as the *advice* and it must adhere to a particular method signature. The advice is inserted around the existing code at a *point-cut*. Point-cuts are defined in one of two ways. Most frameworks use XML and regular expressions to define where the advice should be inserted. This is good for enhancing existing or third-party code whose source code you don't have access to. With the addition of Annotations to Java 5.0 it's now possible to define point-cuts in the code. This is the approach used in EJB 3.0.

The combination of a point-cut and the advice make up the *aspect*. Your code can be enhanced with the aspect at either runtime or compile time. At runtime the classloader works with the framework libraries to manipulate the bytecode based on the XML descriptor or the annotations. At compile time a second compilation step is needed in which the framework's compiler enhances the classes generated by the javac compiler.

Both approaches have their benefits. Compile time avoids the extra startup cost demanded by the runtime approach, which enhances a class when it's first loaded. However, EJB 3.0 implementations use the runtime approach so their aspect framework can be used with standard annotations. This way you don't have to compile your EJBs for every application server.



John Gilbert is a consultant for Number Six Software with 12 years of experience as an architect of service-oriented systems. He holds a master's degree in software engineering from George Mason University and is an avid proponent of Open Source software.

jgilbert@numbersix.com

Four business entities are needed to enable the business process. The company maintains a list of products. Customers create their accounts through the portal and then submit orders with line items.

The data model sufficiently dictates what entity beans will need to be created and the process model provides the rules for the rules engine. However, the actual services need to be defined (Figure 4). Each step in the business process is modeled by a service operation. The operations are then grouped into services based on functional or data cohesion. For example, all the operations that act on Order entities are put in the OrderService.

Several additional components are needed to provide a framework (Figure 5). An Event object is needed to carry information on the bus. A Task entity is needed to store data about the manual steps in the process and a TaskService is defined so users can manipulate the tasks. Finally, a message-driven bean is needed to invoke the rules engine.

Service Implementation

The services provide the actions in the enterprise state machine. Each service is implemented as a Stateless Session Bean by implementing a local and/or remote interface and using the @Stateless annotation (see Listing 1). JSR-181 (Web Services Metadata for Java) will soon let stateless session beans be annotated with a @WebService attribute so an application server can generate WSDL and expose them via SOAP. The @Interceptor annotation defines the point-cut for inserting the EventInterceptor advice around each method of the session bean.

Each of the classes in the business entity class diagram is implemented as a plain old Java object (POJO) and decorated with the @Entity annotation so it can persist. The entity beans will also be used as arguments of the session bean methods. The new POJO nature of the EJB 3.0 entity beans lets them be used outside the container. This simplifies the code necessary to create services because an additional Value object class hierarchy isn't needed to transport the data from the entity beans.

The new EJB 3.0 Entity Manager class is used to persist the entity beans. The @Inject annotation instructs the EJB container to assign the appropriate manager. The manager's persist() method is used to insert the object into

the database. The merge() method is used to update the database.

AOP Interceptor

Interceptors facilitate firing events in the enterprise state machine. AOP interceptors are a kind of advice that's used to insert code around the invocation of a method dynamically and ensure that every service invocation publishes an event to the ESB. The EventInterceptor class takes all the invocation data from the method and puts it in an Event POJO (see Listing 2). An interceptor class must define an invoke() method that's annotated with the @AroundInvoke attribute. Each session bean is annotated with the @Interceptor attribute. (See Listing 1). When each method on the annotated session bean is executed the interceptor is invoked first and passed the invocation context. The interceptor extracts the context information and puts it in the Event POJO. It then tells the method to proceed with its execution. Once the method completes, the output is also put in the event. If an exception is caught then it's put in the event instead and the event name is appended with a Fault suffix to facilitate the error-handling rules. The finally block handles publishing the event to the topic. (Listings 2–4 can be downloaded from www.sys-con.com/java/sourceccfcm.)

ESB Implementation

The rules provide the conditions in the enterprise state machine. To implement the ESB and orchestrate the business process a JSR-94-compliant rules engine is used. I used Drools because of its native support for Java syntax. An MDB is used for the execution environment. (See Listing 3). The @MessageDriven and @ActivationConfigProperty annotations are used to replace the deployment descriptor. The MDB will be activated by a topic destination called topic/net.jcg.EventTopic and its subscription is durable so no events are dropped. The @Inject annotation is used to access the rules engine from the JNDI tree. The onMessage() method extracts the Event object from the message, passes it to the onEvent() method, and gracefully handles exceptions. The onEvent() method uses the RuleRuntime object to create a StatelessRuleSession. A stateless session is used so the session's memory starts fresh for each event. The session is created with the name of the rule set that will be evaluated, "OrderProcess." Once the session is created the rules are

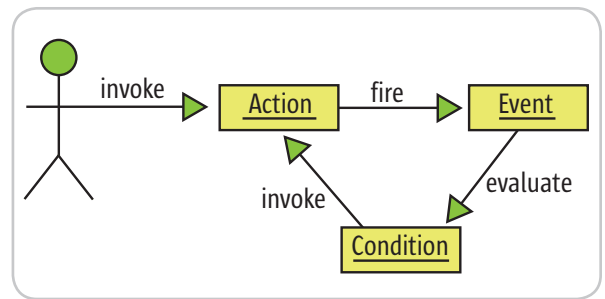


Figure 1 State Machine collaboration diagram

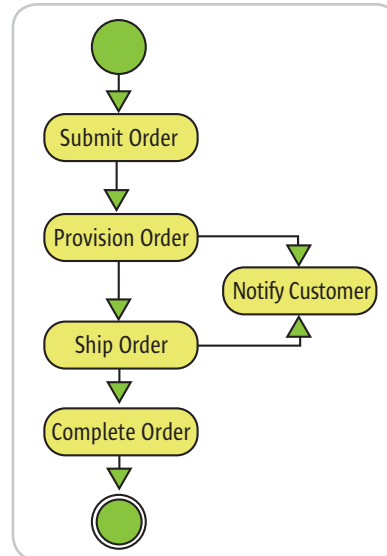


Figure 2 Business process activity diagram

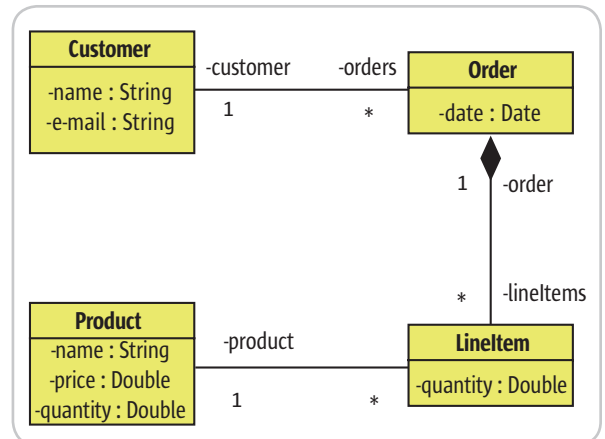


Figure 3 Business entity class diagram

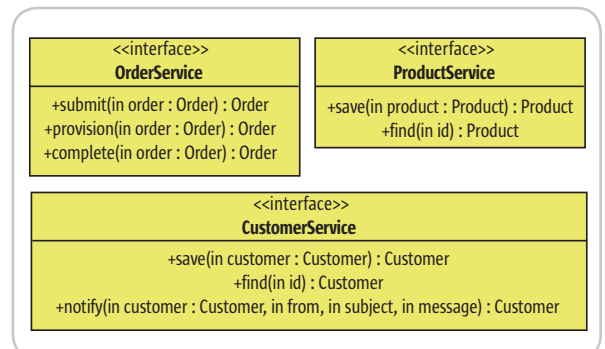


Figure 4 Services class diagram

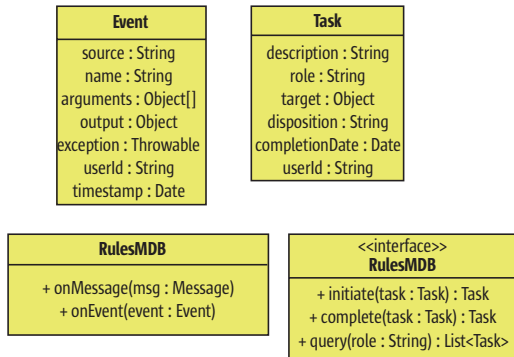


Figure 5 Framework class diagram

executed by passing the event object as input to the executeRules() method.

The rules engine evaluates the rules against the state of the system. The rules are defined in XML and are loaded at startup. The rules are grouped into rule sets. (See Listing 4). Each rule consists of parameters, zero or more conditions, and a consequence. The parameters define the classes in the RuleSession that will be used in the conditions and consequence.

For example, the Event object published by the EventInterceptor will be the root container of all objects used in the rule. Since the objects contained in the event are entity beans their relationships to other objects can be traversed by the rules to retrieve additional system data relevant to evaluating the state of the system. If a rule evaluates to true, then logic is executed that will invoke the next services in the business process. Those services will in turn publish their own events. Since the MDB uses container-managed transactions all the services invoked by rules will be part of the same transaction. Furthermore, the

rules are evaluated asynchronously with regard to the session bean that fired the event. So the service doesn't have to wait for all the rules to complete.

The onSubmitOrder rule tests the source and name of the event. It then extracts the Order from the event and invokes the OrderService provision operation. The onProvisionOrder rule initiates a manual step in the business process by using the TaskService to assign someone with the "Shipper" role to ship the order.

Conclusion

Using this approach, developers can focus on the true intent of their services and a business is free to change the rules to stay in tune with business drivers. The service operations are focused on the atomic unit of functionality that they must provide. They aren't burdened by tedious plumbing code for publishing events to the bus and they are decoupled from the rules about how they might interact with other services.

The architecture is also extensible. Services can be added and removed without breaking other services. Rules can be added, changed, and removed without modifying the services. Any number of JMS subscribers can be added to the bus. Some examples might be: additional rules engines, an AuditMDB that logs all events, an IndexerMDB that indexes all the entity beans contained in an event for access by search engines, or a DataSummarizationMDB that loads a data warehouse in real-time. Legacy services can also be enhanced by AOP to publish to the bus without modifying the code.

The biggest payoff is when this approach is scaled up and plugged into a federated peer-to-peer network of JMS

providers. Each JMS provider broadcasts its presence to the others. Events then flow across the ESB from provider to provider the way e-mail flows through mail servers and packets flow through routers. As a result, services are free to be composed into enterprise-wise business processes without upfront knowledge or modification and without being coupled into a monolithic monster. All you have to do to prepare for this is add a little AOP to your services. ☛

References

- *Demystifying the Enterprise Service Bus* – <http://www.bijonline.com/PDF/Sep03Saini.pdf>
- *12 Steps Toward SOA* – <http://www.bijonline.com/PDF/linthicum%20column%20august.pdf>
- <http://www.bijonline.com/PDF/linthicum%20sept.pdf>
- <http://www.bijonline.com/PDF/Linthicum%20oct.pdf>
- *EJB 3.0 Tutorial* – <http://docs.jboss.org/ejb3/tutorial/index.html>
- *JSR-94 Rules Engine Specification* – <http://www.jcp.org/aboutJava/communityprocess/review/jsr094>
- *Drools Rules Engine* – <http://drools.org/>

Resources

- *Introduction to Aspect-Oriented Programming* – <http://www.onjava.com/pub/a/onjava/2004/01/14/aop.html>
- *JBoss AOP* – <http://www.jboss.org/products/aop>
- *AspectWerkz* – <http://aspectwerkz.codehaus.org/>
- *AspectJ* – <http://eclipse.org/aspectj>
- *Nanning* – <http://nanning.codehaus.org/>

Listing 1: OrderServiceBean

```

@Stateless
@Interceptor("net.jcg.event.interceptor.EventInterceptor")
public class OrderServiceBean implements OrderService {

    @Inject
    private EntityManager manager;

    public Order submit(Order order) {
        order.setDate(new Date());
        manager.persist(order);
        return order;
    }

    public Order provision(Order order) {

```

```

        ...
    }

    public Order complete(Order order) {
        manager.merge(order);

        for (LineItem item : order.getLineItems()) {
            double qty = item.getQuantity();
            Product p = item.getProduct();
            p.setQuantity(p.getQuantity() - qty);
            manager.merge(p);
        }
        return order;
    }
}

```

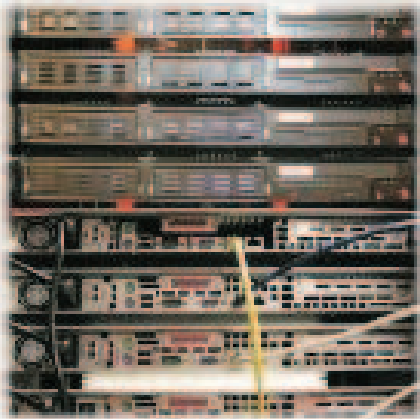
Complex J2EE Hosting made easy.

WebAppCabaretsm

<http://www.webappcabaret.com/jdj.jsp>
1.866.256.7973

\$49
monthly

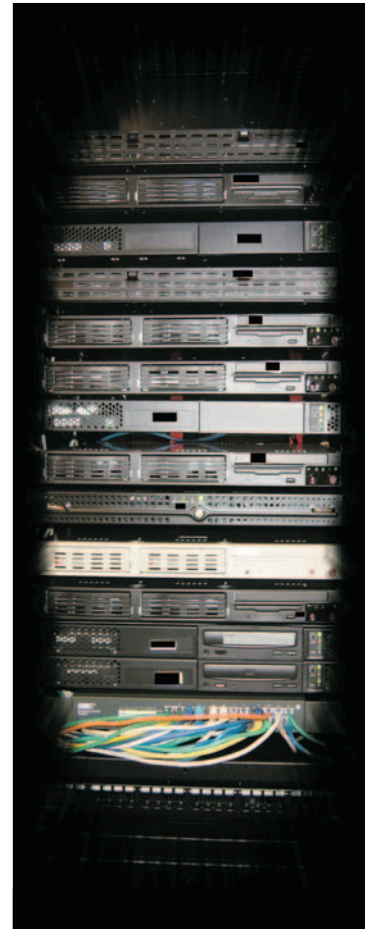
Private JVM
Complete J2EE Hosting solution
for medium size web site.



\$279
monthly

Dual Xeon
2GB RAM
2 x 73GB
SCSI
2000GB
monthly
transfer.

J2EE and
Cluster
ready.



Imagine a hosting company dedicated to meet the requirements for complex web sites and applications such as those developed with Java J2EE.

At WebAppCabaret our standards based process and tools make deploying Java J2EE applications as easy as a point-and-click.

We call it **Point-and-Deploy Hosting**.

Our advanced NGASI Web Hosting management Control was designed for the hosting and management of complex web sites and applications thus cutting down on maintenance time.

Backed by an experienced staff as well as a **Tier 1 Data center** and network. Our network is certified with frequent security audits by reputable security firms.

All hosting plans come with **advanced tools** to manage your application server and Apache web server. Not to mention the other features, such as virus-protected email, bug tracking, and many more portal components.

Complete power and control at the tip of your fingers. We take care of the system and hosting infrastructure so you can concentrate on development and deployment of your application. That is **real ROI**.

Log on now at <http://www.webappcabaret.com/jdj.jsp> or call today at 1.866.256.7973



WebAppCabaretsm

Point-and-Deploy J2EE Hosting

Prices, plans, and terms subject to change without notice. Please log on to our website for the latest price and terms. Copyright © 1999-2005 WebAppShowcase • All rights reserved • Various trademarks held by their respective owners.



Calvin Austin

Core and Internals Editor

What's in a Name: Is This the End of J2EE?

There has been talk recently that Sun is planning to end the use of the J2 platform name and branding scheme. The proposal is that the terms *Java Enterprise Edition* and *Java Standard Edition* will replace J2EE and J2SE. If you had the opportunity to read the March edition of *JDJ*, you will have discovered that the history of the J2 name, although not perfect, was a compromise by recognizing the platform had taken a significant step forward and resisting the temptation for calling the new release Java 2000.

Given that history, a name change was probably always on the cards but there was never a good time to make that change. In the early days of J2EE, getting vendors to adopt and use the term J2EE was essential for the success of the platform as an industry-wide event. Changing the J2EE name at that point would have confused matters to no end, so a name change was a nonstarter. Later, with the emergence of the .NET Framework from Microsoft, the Java community rallied behind the J2EE banner when comparing the two platforms. Microsoft even used this comparison, partly in an attempt to validate a very new .NET platform, even to the extent of focusing less on the comparison of Java versus C#.

Why change one of the most successful brands now? Only the marketing teams at Sun know the real answer. The J2SE name in particular never got the industry recognition that J2EE received, but changing all the brands for consistency is never cheap. What it means for the average Java developer is unclear. It is now more descriptive to say that you are a Java Enterprise Edition developer, but at the same time clients and recruiters will still be looking for a J2EE developer so I doubt the J2EE name will ever really disappear.

Regardless if you call yourself a J2EE or Java Enterprise Edition developer, there were two other pieces of interesting news this month. The first was another article from Sun about the Barcelona project (the Multi-Tasking Virtual Machine, MVM); the second item was the first shipping Java system from Azul Systems.

The Multi-Tasking Virtual Machine

The MVM JVM is still a research project but garners a lot of attention in and outside Sun. The claims are that using the MVM technology will result in more scalable JVMs with improved startup time. The version I tried at Sun didn't use the standard

spin off applications that would be a higher risk, for example, using JNI in their own process space. We will have to wait to see a public download, but getting a product from research into production is hard and requires a lot of dedication.

A New Java CPU

The first shipping Java appliance from Azul Systems is the second piece of news of note. In the early days of Java there was some interest in having a Java CPU; however, the focus was to replace the most complex application of all, the desktop. Azul Systems have mitigated this by narrowing their scope to J2EE application servers. However, the big surprise is the cost of a dedicated Java CPU.

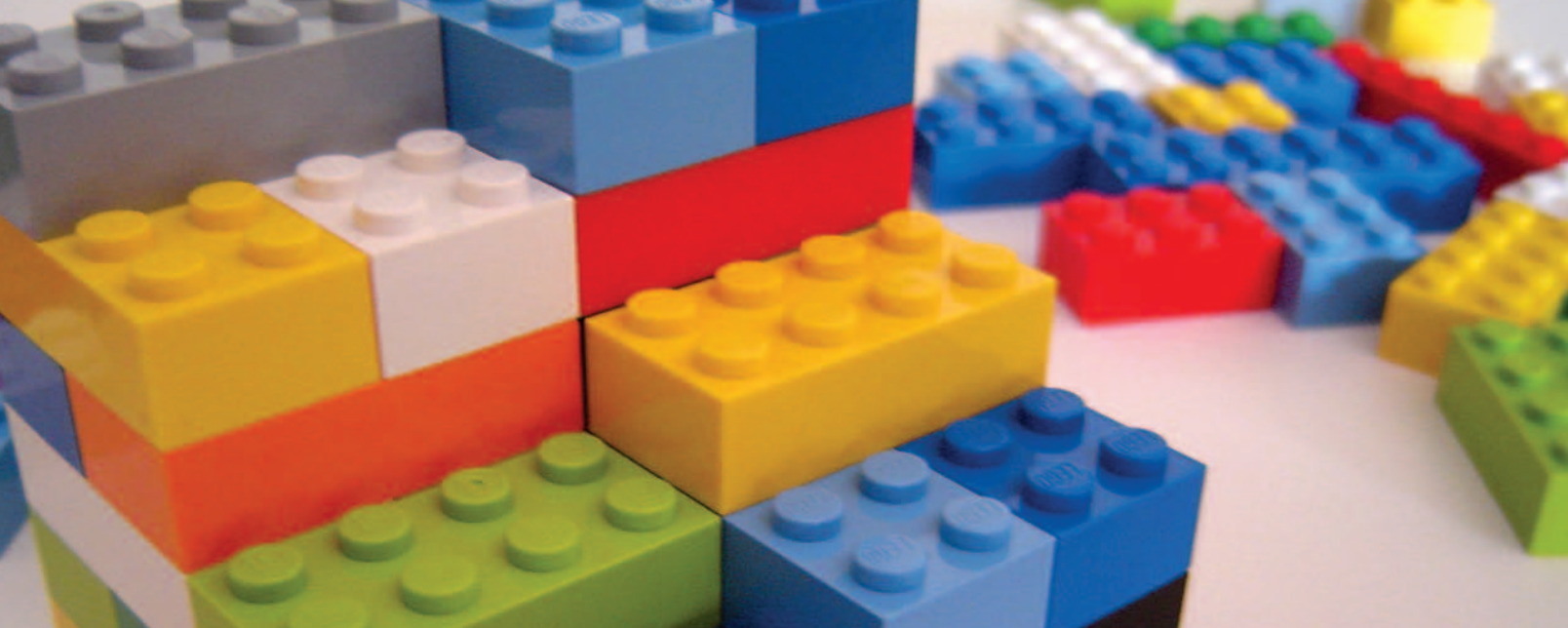
CNET reports that the base production systems cost just under \$200,000 for a 192 CPU system, or approximately \$1,000 dollars per processor and peaks out at just under \$800,000 dollars. That is a great leap of faith for some, although the price is in-line with large systems from traditional Unix vendors. There is a trial 89 CPU system for demo users but time will tell if there is demand for such a powerful system. My own employer's Web site delivers most of its Java application services through a cluster of four machines and two database servers, and even when we first started the Java developer connection site we literally had a single ultra 1 machine. Yet it's a testament to Java technology that no one is rushing to build a C# or .NET processor to compete.

In closing, next month's edition of *JDJ* will be focused on JavaOne, probably still the largest Java conference in the world. If you have any interesting stories to share please let me know at calvin.austin@sys-con.com. I have been fortunate to have attended every show to date and will be sharing my tips and tricks on making the most of the conference. ☛



isolated JSR and I didn't get very far making it work on Solaris. The MVM model is similar to other service style models like the X-based console/terminal service. The X-based console service does not start a new process each time a terminal request is received but instead spins off a new instance. The huge advantage is decreased startup time; the traditional downside is that if one console experiences an error, it can take down all the other running consoles – something that was very frustrating for early users. The MVM plan is to

A section editor of *JDJ* since June 2004, Calvin Austin is an engineer at SpikeSource.com. He previously led the J2SE 5.0 release at Sun Microsystems and also led Sun's Java on Linux port. calvin.austin@sys-con.com



Do it yourself.

Mobilize your business without losing money or sleep.
Mobilize overnight. It's easy, manageable and fast.



CTIA WIRELESS 2005

A Division of CTIA - The Wireless Association™

Come visit us at Booth #6948

Phoneomena products allow the enterprise to easily create its own mobile applications or mobilize existing applications without having to learn new technologies or make upfront investments. With Phoneomena products you will not be locked into any mobile platform and will not waste time and resources learning J2ME, Brew, Windows Mobile, Palm OS or Symbian. You will be freed to focus on your mobilization strategy to achieve your business goals. You will literally be able to mobilize overnight, using only the standard web knowledge your IT team has today.

Phoneomena, Inc.
104 N. Main St Suite 300
Gainesville, FL 32601

T: +1 (352)-373-3966
F: +1 (352)-373-3651
www.phoneomena.com

Toll-free: +1(888) 891-7316
Email: info@phoneomena.com

PHONEOMENA

Understanding JSSE

by Sudhir Upadhyay

The Java Secure Socket Extension (JSSE) is a set of packages that enable secure Internet communications. It implements a Java version of the Secure Sockets Layer (SSL) and Transport Layer Security (TLS) protocols. It includes functionality for data encryption, server authentication, message integrity, and optional client authentication.

Although the JSSE guide provides detailed information on the JSSE API and its use in application programming, this article dives deeper into the different message exchanges involved when a programmatic Java client communicates with a server over the SSL. It will help developers understand the fundamental but often overlooked JSSE concepts of keystore, truststore, cipher suites, certificates, and the public key infrastructure and help them solve some of the common programmatic and configuration issues that arise when developing Java clients that communicate over SSL. It enhances the JSSE guide documentation by walking the reader through a debug output and explaining each message exchange between a client and the server in detail. Before starting, it's important to note that JSSE is fundamentally different from the Java Authentication and Authorization Service (JAAS).

Secure Socket Layer

The SSL Handshake Protocol was developed by Netscape Communications Corporation to provide security and privacy over the Internet. SSL ensures that the communications between two parties is secured and encrypted. It ensures that an intruder on the network can't see the data exchange between a sender and receiver or decipher it in any form. SSL does this by fragmenting the message into multiple manageable blocks, applying a Message Authentication Code (MAC), encrypting the message, and finally transmitting it to the intended party.

One of the common uses of SSL is in Internet commerce. While Internet browsers have used SSL for several years now, its operation remains transparent to most users because the browsers have the in-built capability of creating a SSL socket to the server and handling all the underlying complexities.

Cryptography

One important concept used in SSL is cryptography. Cryptography is essential for the secure exchange of information across intranets, extranets, and the Internet. From a business point of view, the security functions enabled by cryptography are *authentication*, which assures the recipient of a message that the originator is who he or she claims to be; *confidentiality*, which ensures that a message can be read only by the intended recipient; and *integrity*, which assures the recipient that a message hasn't been altered in transit.

From a technical point of view, cryptography is the science of protecting data by mathematically transforming it into an unreadable format. For a typical Web user, the browser does this. In other words, when a user types a secure Web site in the URL (<https://www.mysecuredsite.com>), the browser ensures that server is who the server says it is. The server's authenticity is verified via the digital certificate presented by the server.

Public Key Infrastructure

Although any detailed description of cryptography is beyond the scope of this article, since Public Key Infrastructure (PKI) is a commonly applied cryptography technique in most current SSL implementations, it may be a good idea to touch on PKI briefly.

PKI is based on two keys – a *public key* and a *private key* – that are mathematically related, and are used in public key encryption. In public key encryption, the public key can be passed publicly between the parties or published in a public repository, but the associated private key remains private to the owner of the certificate. Data encrypted with the public key can be decrypted only with the private key and conversely data encrypted with the private key can be decrypted only with the public key. However, since the public key is publicly available, it would be naïve to encrypt data using the private key.

In browser-to-server communication over SSL, when the browser connects to the server, the server sends its certificate to the client. The server certificate contains, among other things, the server's public key.

In figure 1, when the browser connects to a secure site, the browser requests the server's certificate, which includes the server's public key. When it gets the server's certificate, the browser performs the following steps:

- Validates the certificate based on its authenticity and expiry.

Sudhir Upadhyay is a principal consultant with BEA professional services where he helps customers in designing and implementing enterprise applications. He is a Sun Certified Java Programmer and has been working exclusively in Java since 1998.

sudhir@bea.com

- Uses the certificate it gets to generate its own private/public key pair
- Uses the public key from the server certificate to encrypt the data to send to the server.

At the end of this handshake, the browser has three keys:

- The server's public key – used to encrypt messages sent to server
- Its public key – to send to the server
- Its own private key – to decrypt messages sent by the server that are encrypted by browser's public key.

When the handshake is over, both the browser and the server are ready to communicate with each other over a secure channel. However, all the above message exchange between the browser and the server remains transparent to the user of the site. The user only has to tie in the URL of the server he wants to communicate with. The browser handled all of the message exchange, key generation, and SSL handshake. However, when a programmatic client connects to the SSL server, the client's developer needs to do a few configuration steps before it can communicate with a server over SSL.

With this brief introduction, let's get down to the details of each step a programmatic Java client follows when connecting to a server.

Client Configuration

Although developing a secured socket connection isn't significantly different from conventional socket connection programming, understanding the underlying message exchange is valuable in troubleshooting any issues that may arise.

As described above, the asymmetric key cryptography PKI requires that both parties involved in data exchange send each other their public keys. So, for a programmatic client to communicate with a server, the private/public key pair has to be generated and made available to the client program at runtime. Like a browser, the programmatic client's private/public key pair is derived from the server's certificates. However, since the server certificate is sent only when the client connects to the server, the client needs to get the server certificate by other means before initiating a SSL connection.

There are a couple of ways a client can access the server certificate. A quick and easy way is to use the browser's ability to import server certificates and copy that to the file. In other words, if the server that has the certificate hosts any Web application that can be accessed by an Internet browser over a SSL connection, you may be able to import the certificate from, say, Internet Explorer. (For a detailed description of how to import the server certificate from the browser, you should refer to the browser-specific help.)

The other option generally followed in the industry is to manually distribute the public key (certificate) of the server to the programmatic clients. Generally, the server certificates are in *.PEM, *.DER, or *.CER formats and typically contain information such as issuer, valid dates, signature algorithm (MD5, RSA), subject, and the public key of the server.

Once the certificate has been obtained from the server, it's stored in a keystore on the client machine so it's accessible by the client at runtime. In a JRE model the keystores primarily comprises two files: a "keystore" and a "truststore." The keystore consists of an entity's identity and its private key. The keystore



DynamicPDF™ Merger v3.0

Our flexible and highly efficient class library for manipulating and adding new content to existing PDFs is available natively for Java

- ♦ Intuitive object model
- ♦ PDF Manipulation (Merging & Splitting)
- ♦ Document Stamping
- ♦ Page placing, rotating, scaling and clipping
- ♦ Form-filling, merging and flattening
- ♦ Personalizing Content
- ♦ Use existing PDF pages as templates
- ♦ Seamless integration with the Generator API

DynamicPDF™ Generator v3.0

Our popular and highly efficient class library for real time PDF creation is available natively for Java

- ♦ Intuitive object model
- ♦ Unicode and CJK font support
- ♦ Font embedding and subsetting
- ♦ PDF encryption ♦ 18 bar code symbolologies
- ♦ Custom page element API ♦ HTML text formatting
- ♦ Flexible document templating

Try our free Community Edition!



DynamicPDF™ components will revolutionize the way your enterprise applications and websites handle printable output. DynamicPDF™ is available natively for Java.



is used for various cryptographic purposes. The truststore contains the public key, in addition to the entity's identity. A *truststore* is the keystore used when making decisions about what to trust. If you get some data from an entity that you already trust, and if you can verify that the entity is the one it claims to be, then you can assume that the data really came from that entity. Combined together these files are generically referred to as **keystores**.

The files are stored/imported in the keystore via a tool provided by JRE. In JRE, this tool is called a keytool and is available with the JRE distribution. For example:

```
keytool -import -alias myAlias -file CertGenCA.pem -keypass mypasswd
-keystore securekeystore - storepass mypasswd
```

This command creates the file name *securekeystore* and imports the certificate from the file *CertGenCA.pem* with the alias *myalias*. It also checks the validity of the certificate and ensures that the certificate hasn't expired. (For additional help on keytool, see the keytool documentation at <http://java.sun.com/j2se/1.4.2/docs/tooldocs/windows/keytool.html>)

Programming Details

After the server certificate is imported into the keystore and is accessible by the client, the client is ready to communicate with the server. The complete client code can be found in Listing 1.

The following section will explain the significance of the different programming steps and what really happens under the layers.

Initiating the SSL connection is fairly straightforward and quite similar to conventional socket programming. In fact, there are only two APIs (line number 13 and 14 in the code listing) that are different in a SSL communication from a plain socket connection. So the remaining section will walk you through the sequence of activities involved while invoking these two APIs.

SSLFactory Implementation

The first step in socket programming is obtaining a reference to the *SocketConnectionFactory* as indicated below. (The text in green is what a developer writes and the text in blue represents the debug output obtained by running the client.)

```
SocketFactory sf = (SocketFactory)SocketFactory.getDefault();
```

This command returns a copy of the environment's default socket factory. All subsequent objects, such as *Socket*, are derived from the socket factory.

In a similar manner, the first programming step in initiating a secured socket connection is obtaining a reference to *SSLSocketFactory*.

```
SSLSocketFactory sslSF = (SSLSocketFactory)SSLSocketFactory.
getDefault();
```

This returns the default SSL implementation, i.e., policies for different kinds of socket, and any customizations on how they are configured.

Let's walk through the debug output from running the client program:

```
java -Djavax.net.debug=ssl -Djavax.net.ssl.KeyStore=secureKeyStore
-Djavax.net.ssl.trustStore=secureKeyStore SecureSocketClient
```

It's important to note that all the steps explained below are transparent to the developer and don't require any additional programming effort. The following is the sequence of processes performed when the client programs creates an *SSLSocketFactory*.

When initializing the *SSLDefaultSocketFactory*, the underlying implementation provides the following services. The debug output is in *italics*.

The SSL implementation identifies the keystore location as specified by the system java property *-Djavax.net.ssl.keystore* when the client is invoked. The type of keystore. If no value was specified, then the default *KeyType* is "jks" is assumed.

```
keyStore is : D:\\secureKeyStore
keyStore type is : jks
init keystore
```

Similarly, the location and the type of *trustStore* (*-Djavax.net.ssl.trustStore*) are associated with the *SSLContext*.

```
trustStore is: D:\\Securekeystore
trustStore type is : jks
init truststore
```

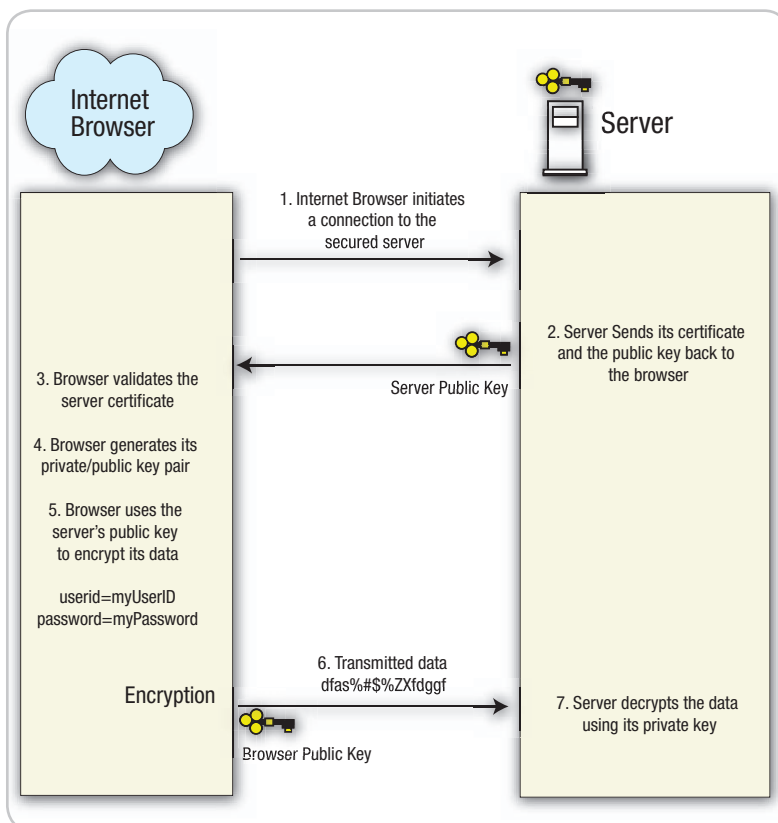


Figure 1 Browser-server communication

Identify the *KeyManager* and *TrustManagers*. A *TrustManager* is used to authenticate the remote identity of a secure socket peer. A *KeyManager* is used to authenticate a local socket peer to a remote secure socket peer. A default *SSLContext* is initialized with a *KeyManager* and *TrustManager*.

init keymanager of type SunX509 reads all the available certificates from the trust and loads it into memory.

```
adding as trusted cert: [
[
  Version: V3
  Subject: CN=CertGenCAB, OU=FOR TESTING ONLY, O=MyOrganization,
  L=MyTown, ST=MyState, C=US
  Signature Algorithm: MD5withRSA, OID = 1.2.840.113549.1.1.4
```

SSL requires a cryptographically secure pseudo-random number generator (PRNG). With a pseudo-random number generator, given a short random input (the key), one can generate an output string as long one needs and is impossible to distinguish from a truly random string. In Java the *SecureRandom* class provides this capability. During this step, the SSL implementation instantiates a *SecureRandom* class and seeds it for use later.

Socket Creation Implementation

So far we have only discussed the steps involved on the client side before the socket connection to the server is initiated.

The next step in client-to-server communication is to create the socket connection as indicated below:

```
SSLSocket sslSock = (SSLSocket) sslSocketFactory.
createSocket(host,port);
```

After a successful invocation of the above call, the mutual handshake between the client and the server is considered complete. Both the client and the server are then ready to exchange the data over a secure channel.

The following section explains the sequence of message exchanges during the mutual SSL handshake. Again, it's important to remember that these message exchanges remain transparent to the developer.

Hello Messages

The Client and the Server Hello requests constitute hello messages.

Client Hello

The client sends the following client "hello" message to the server as indicated by the following debug output:

```
[java] *** ClientHello, TLSv1.
```

The purpose of the "Hello" message is to establish enhancement capabilities between the client and the server. In other

The World's Leading Java Resource Is Just a >Click< Away!



JDJ is the world's premier independent, vendor-neutral print resource for the ever-expanding international community of Internet technology professionals who use Java.

Only **\$69⁹⁹** ONE YEAR 12 ISSUES

Subscription Price Includes **FREE** JDJ Digital Edition!

www.SYS-CON.com/JDJ
or **1-888-303-5282**

OFFER SUBJECT TO CHANGE WITHOUT NOTICE

words its purpose is to agree on algorithms, exchange random values, and check for session resumption, if there's already an established session.

The hello message typically contains a random structure that's used later in the protocol. The random structure contains the current time and date in standard Unix 32-bit format and 28 bytes generated by the secure random generator. This is the same SecureRandom class that was instantiated during the SSLSocketFactory implementation. The following represents an example of a random structure.

```
[java] RandomCookie: GMT: 1068318819 bytes = {179, 149, 240, 59, 32,
133, 114, 223, 214, 179, 158, 252, 216, 163, 195, 81, 38, 109, 86,
103, 87, 233, 180, 113, 250, 85, 224, 249 }
```

The client hello message also includes a variable length session identifier. This is the identity of the session corresponding to the connection. Generally, this field is empty when no session_id is available as shown below, or the client is initiating a new session and wants to generate new security parameters.

```
[java] Session ID: {}
```

The next field in the *Hello* message is the cipher suites supported by the client. The client sends the list of all the cipher suites it supports. This is the list of cryptographic algorithms supported by the client in the order of the client's choice (first choice first). Each CipherSuite defines a key exchange algorithm, a bulk encryption algorithm (including secret key length), and a MAC algorithm. The server will select a cipher suite (usually the first in the list, if supported by the server) or, if no acceptable choices are presented, return a handshake failure alert and close the connection.

The following is an example of the cipher suites sent by the client to the server.

```
[java] Cipher Suites: [SSL_RSA_WITH_RC4_128_MD5,..... TLS_DHE_
RSA_WITH_AES_, SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA]
```

Along with the CipherSuites, the client also sends a list of the compression methods supported by the client, sorted by client preference. The compression algorithm translates an SSLPlainText structure into an SSLCompressed structure. This algorithm may be used to compress the data prior to encryption. In the following debug output, the compression method.

```
[java] Compression Methods: { 0 }
```

In essence, during the hello message, the following security attributes are established.

- *Protocol Version*: The version of the SSL protocol by which the client wishes to communicate during this Session and is typically the highest valued version supported by the client.

In the above example, the protocol version is TLSv1.

- *Session ID*: Used to identify any existing session between the same client and the server.
- *Cipher Suite*
- *Compression Method*

In response to the above Hello request from the client, the server can either send a Hello message back or return with a handshake_failure alert. The following sections explain the messages on the server side in response to the Hello Request from the client.

Server Hello

The server will send this message in response to a client hello message when it finds an acceptable set of algorithms. If it can't find a match, it will respond with a handshake failure alert. The following is the debug output from the Java client:

```
[java] *** ServerHello, TLSv1
```

Just as the client generated a random structure, the server also generates a random structure to be used later in the protocol. The random structure is similar to the one shown below.

```
[java] RandomCookie: GMT: 1068318819 bytes = { 152, 122, 230, 150,
51, 26, 74, 27, 140, 113, 192, 13, 22, 76, 228, 17, 150, 251, 234,
30, 155, 33, 77, 179, 30, 31, 60, 155 }
```

The server then generates a session id for the connection. If the client session id wasn't empty, the server will look in its cache to find a match. If a match is found, the server can reuse this session, otherwise it will generate a new session id (as shown below).

```
[java] Session ID: {254, 66, 184, 128, 0, 67, 156, 252, 120, 36,
183, 89, 192, 173, 85, 191}
```

If the server sends an empty session id back to the client, it implies that the server doesn't want to re-use the existing sessions.

In the next step, the server traverses the list of cipher suites supported by the client and selects the highest supported between the client/server. It also selects the available compression algorithm from the list sent by the client. As you'll see in the following output, the server has picked the following Ciphersuite (the first in the list sent by the client):

```
[java] Cipher Suite: SSL_RSA_WITH_RC4_128_MD5
And the compression method, again from the client list.
[java] Compression Method: 0
```

If either the ciphersuite or the compression method does not match, the session will fail.

```
[java] %% Created: {Session-1, SSL_RSA_WITH_RC4_128_MD5}
```

“Cryptography is essential for the secure exchange of information across intranets, extranets, and the Internet”

Certificate Exchange

In its response to the client request, the server sends a certificate. Although this step can be optional, in most cases it's mandatory because in typical SSL implementations the client invariably requests the server's authentication. In an anonymous key exchange, none of the parties (neither the client nor the server) are authenticated.

This mode is vulnerable to a man-in-middle attack and isn't a widely used key exchange mode. The following is a sample of a server certificate key exchange:

```
[java] *** Certificate chain
[java] chain [0] = [
[java] [
[java]   Version: V1
[java]   Subject: CN=username, OU=FOR TESTING ONLY, O=MyOrganization,
L=MyTown, ST=MyState, C=US
[java]   Signature Algorithm: MD5withRSA, OID = 1.2.840.113549.1.1.4
```

The server certificate will always immediately follow the *server hello* message. Once the certificate message is sent from the server to the client, the *server hello* is over.

```
[java] *** ServerHelloDone
```

Key Exchange

Once the server has sent a certificate to the client, a Key Exchange message is initiated. It is always sent by the client. It will immediately follow the client certificate message, if it's sent. Otherwise it will be the first message sent by the client after it gets the server hello done message.

Client Key Exchange Message

```
[java] *** ClientKeyExchange, RSA PreMasterSecret, TLSv1
```

The purpose of the key exchange process is to get a *token encryption key* (TEK), which is used to wrap data encryption keys, client write keys, server write keys, and master secret encryption keys. The encrypted *pre_master_secret* is sent to the server in a client key exchange message. The *pre_master_secret* will be used to generate the *master_secret*. The *master_secret* is needed to generate the finished messages, encryption keys, and MAC secrets. By sending a correct finished message, the parties prove that they know the correct *pre_master_secret*.

Finished Message

The last message in the SSL handshake process is the "finished message." A finished message is always sent immediately after a change cipher spec message to verify that the key exchange and authentication processes were successful. It's essential that a change cipher spec message be received between the other handshake messages and the finished message. The following output demonstrates a finished message exchange.

```
[java] *** Finished
[java] verify_data: { 155, 104, 57, 51, 140, 66, 235, 165, 133, 234, 48,
234 }
[java] ***
[java] main, WRITE: TLSv1 Handshake, length = 32
[java] main, READ: TLSv1 Change Cipher Spec, length = 1
```

```
[java] JsseJCE: Using JSSE internal implementation for cipher RC4
[java] main, READ: TLSv1 Handshake, length = 32
[java] *** Finished
[java] verify_data: { 237, 42, 200, 45, 111, 152, 20, 147, 77, 110, 221,
199 }
[java] ***
```

The finished message is the first protected message with the just-negotiated algorithms, keys, and secrets. Once a side has sent its finished message and gotten and validated a finished message from its peer, it may begin to send and receive application data over the connection. At this point, the SSL handshake has been established and both the client and server are now ready to exchange data over a secured connection.

Summary

This article provided you with the message exchange between a Java client and a server when communicating over a secured socket layer, as implemented by JSSE. The multiple message exchange between the client and the server are crucial to establishing the identity of the each party, agreeing on different cryptographic algorithms, protocol version, compression, and encryption methods. 🍌

References

- Java Secure Socket Extension Guide <http://java.sun.com/j2se/1.4.2/docs/guide/security/jsse/JSSRefGuide.html>
- Cryptography FAQ http://www.rsa-security.com/rsalabs/faq/files/rsalabs_faq41.pdf

Listing 1: Sample Java Client

```
import java.io.*;
import javax.net.ssl.SSLSocket;
import javax.net.ssl.SSLSocketFactory;

public class SecureSocketClient {

    public static void main (String [] argv) throws Exception
    {
        try {
            //SocketFactory sslSF = (SSLSocketFactory)SSLSocketFactory.getDefault();
            //Socket sock = (SSLSocket) sslSF.createSocket("localhost",443);

            SSLSocketFactory sslSF = (SSLSocketFactory)SSLSocketFactory.getDefault();
            SSLSocket sslSock = (SSLSocket) sslSF.createSocket("localhost",443);

            OutputStream out = sslSock.getOutputStream();
            String req = "GET /Secure.jsp HTTP/1.0\r\n\r\n";
            out.write(req.getBytes());
            BufferedReader in = new BufferedReader( new
            InputStreamReader( sslSock.getInputStream()));
            String inputLine;
            while ((inputLine = in.readLine()) != null)
                System.out.println(inputLine);
            sslSock.close();
        } catch (Exception _ex){
            _ex.printStackTrace(System.out);
        }
    }
}
```



Joe Winchester
Desktop Java Editor



Total Eclipse

Tim O'Reilly, the eponymous publisher, kicked off EclipseCon 2005 in Burlingame earlier this year with an excellent presentation titled "Open source business models and design patterns." As well as documenting various failures and successes in the computing world, one message that struck a chord was that to succeed in open source you must design for participation. Three days later, Lee Nackman, CTO of IBM Rational Software and one of the original thinkers behind the Eclipse project, demonstrated how this was one of the core principles built in from the ground up. It had led to the creation of the independent Eclipse Foundation that, among its other goals, had the aim of achieving serious participation from more vendors. This was demonstrated more than anything at the conference that, while last year was being talked about as a concept, occurred this year.

BEA, Borland, Scapa, and Sybase recently became strategic members of the Eclipse Foundation and are driving forward key projects. A quick look at the list of technology projects on <http://www.eclipse.org/technology/> shows the level of participation and diversity of the efforts. The number of projects that were newly announced in the past few months is a nice litmus indicator of the health of Eclipse and its current growth phenomenon.

The booth floor of the conference had a large number of companies showcasing everything from testing and reporting products through code analyzers and high-level design tools. In 2004 many exhibitors were touting Eclipse integration while showing their existing product line elegantly shoe-horned into the workbench. This seems to be less prevalent where everyone from mom and pop to the big guns of software were demonstrating products that had a very high degree of fit and finish with respect to the platform and its integration API.

In 2004 there was a buzz around the ideas behind the Rich Client Platform (RCP); this year these were being demonstrated as a very real and working technology. The RCP's idea is that folks building tools for Eclipse should not be the only ones to benefit from its plug-in architecture, and developers of end-user applications in any domain can enjoy its concepts such as window management, UI frameworks, and having an update manager. RCP is an odd framework because on first inspection it doesn't really add anything to Eclipse; instead it allows for all vestiges of IDE-like features to be removed so the workbench becomes stripped down to its bare bones. What is impressive with RCP is that this idea has not only worked technically but has become an execution reality with projects such as NASA's Mars rover mission, which is planning on using it. The RCP team has also extended the plug-in development environment (PDE) to include specific wizards to help build RCP while the Visual Editor's SWT GUI builder supports creating RCP views. RCP often reminds me of the Internet, where a relatively simple idea is ready at the right time, free to use, and can have an explosive effect way beyond its initial conception on software development.

One of the questions at the end of EclipseCon 2004 was why the Web Tools Project (WTP) seemed to have stalled. Since then a lot has occurred with WTP and this year's conference had a number of technical sessions and a BoF on WTP showing its progress and the health of the project.

The Eclipse Test and Performance Tools Platform Project (TPTP) is another success story, with the goal of providing a common API and data model for monitoring, testing, tracing, and profiling tools. What is nice about TPTP is the way that its contributors have managed to create a very functional base yet they're the very same

companies that provide value-add in their commercial offerings by extending TPTP in their own product lines that benefit from the core frameworks and data interchange model.

EclipseCon 2005 also saw interest in the Business Intelligence and Reporting Tools (BIRT), which has the goal of providing a set of reporting features and scripting to allow J2EE Web apps to produce Web- and PDF-based reports.

What I find refreshing about BIRT is that it was proposed by Actuate and TPTP is led by SCAPA, underlining the fact that the Eclipse Foundation enjoys and benefits from the depth and breadth of knowledge of the large number of companies that form its base.

I came away from this year's conference with the feeling that Eclipse, by its critical mass of participants, is being propelled along new and interesting paths that continue to stretch and test the framework along its underlying principle of extensibility around a plug-in architecture. One of the slides that stuck in my mind was by Carl Zetie, an analyst with Forrester's Application Development & Infrastructure Research Group. While doing market research on IDE usage, they had asked companies what their adoption of Eclipse was and received two different answers. The reported Eclipse usage from IT managers was lower than the programmers. The grass roots programmers who perhaps didn't like their department's set of tools had adopted Eclipse to become more productive, in a way that reminded me very much of the early growth of Microsoft Windows where individuals installed it and became productive while their IT departments were busy rolling out late and inflexible corporate systems. The fact that Jason Weber from Microsoft gave a talk at EclipseCon encouraging tools providers to develop for both Visual Studio and Eclipse simultaneously adds its own ironic chord. ☺

Joe Winchester is a software developer working on WebSphere development tools for IBM in Hursley, UK.

joewinchester@sys-con.com



INNOVATE

with the Power of Java™

Connect with the full power of Java™ technology at the JavaOne™ conference.

In-depth EDUCATION

Evolve your skills in 300+ of expert-led technical and Bof sessions.

Real-world INNOVATION

Evaluate proven tools and technologies in the JavaOne™ Pavilion.

Global COMMUNITY

Celebrate the tenth anniversary of Java technology.

Visionary INSIGHT

Hear what the future holds from industry leaders.

June 27–30, 2005

JavaOne™ Pavilion: June 27–29, 2005
Moscone Center, San Francisco, CA

Experience a week unlike any other

EXPERIENCE THE 10TH ANNUAL JAVAONE™ CONFERENCE.

Core Platform | Core Enterprise | Desktop | Web Tier | Tools | Mobility and Devices | Cool Stuff

Save \$100!

REGISTER

by June 27, 2005 at java.sun.com/javaone/sf.

JavaOne™

Sun's 2005 Worldwide Java Developer Conference

Copyright © 2005 Sun Microsystems, Inc. All rights reserved. Sun, Sun Microsystems, the Sun logo, Java, the Java Coffee Cup logo, JavaOne, the JavaOne logo, Java Developer Conference, Java Community Process, JCP, 100% Pure Java, J2EE, J2ME, J2SE, Jini, Solaris, "Write Once, Run Anywhere," and all Java-based marks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

SQL Compiler (for Java)

by Pavel Vlasov

Free your Java code from SQL statements – compile them to Java classes

This article describes a SQL Compiler tool (SQLC) that generates Java classes from SQL statement and table metadata. In doing so SQLC decouples Java and SQL code and enforces a clear separation of concerns between database and Java code and division of labor between data modeler and Java developer. The article also ruminates about SQLC, O/R mapping frameworks (taking Hibernate as an example) and plain JDBC applicability in different contexts.

Introduction

Recently I worked on a Java application and needed to do a lot of database interaction. I didn't want to use plain JDBC for obvious reasons (that are nonetheless described below). I had already created a data model and didn't want to do double work by mirroring the data model in Java by hand.

So I looked around for a carpal tunnel friendly tool, something like the erstwhile FoxPro 2.6. To my surprise I found none. Nowadays everybody seems to care about flexibility but not simplicity.

After pondering for a while I preferred creative typing over mundane and wrote a tool that generates interfaces and methods from database metamodel. Lo and behold SQLC was born!

SQLC generates Java classes and interfaces from information provided by `java.sql.Connection.getMetaData()`, `java.sql.PreparedStatement.getMetaData()` and `java.sql.PreparedStatement.getParameterMetaData()` methods. It uses BCCEL and helper classes and techniques described in XREF:ARTICLE_6207. SQLC-generated classes use Squirrel as a foundation, which minimizes the amount of generated code (see resources, don't mix up with Squirrel SQL client).

You may think: "Oh, yet another O/R mapping tool, why would I waste my time reading about it?" Well, SQLC is not an O/R mapping tool at all. Like spiritualists and materialists have different answers to the question "What is primary – spirit

or matter?" SQLC and O/R mapping tools offer different answers to the question "What is primary – data or objects?" SQLC treats data as primary and only as data – i.e. no behavior. For example, a credit card has no behavior per se – it's just a tuple. Behavior belongs to the objects operating with the credit card, not the credit card itself.

SQLC also does something what other tools don't do – it compiles parameterized statements to Java methods.

This is a short summary of SQLC's advantages:

- **Simplicity:** SQLC requires little development time configuration and no deployment time configuration (no deployment descriptors at runtime).
- **Robustness:** Build time verification of SQL statements. SQLC uses the metadata information provided by the target database. Problems with SQL – syntax errors, invalid DB object names – are revealed at build time.
- **Reusability:** Compiled classes can be used by many applications working with the same database. Generated classes, for example, can be distributed as jar files. In application server environments engine classes can be mounted to the JNDI tree. Storing

statements in the database allows you to reuse tuned statements in non-Java applications as well.

- **Separation of concerns and division of labor:** A data modeler models the database (DDL) and develops SQL statements (DML) optimal for the database. A Java developer uses generated classes/methods to access/modify data. He or she doesn't need to know SQL at all and only needs a superficial understanding of the underlying database structure, which will come from SQLC generated documentation.

Corollaries of the previous point are:

- **Modifiability and maintainability:** Compiled classes define an interface between the database and Java code. Data modeler is free to modify DDL and DML – replace joins with nested selects, or include an execution plan to queries – without touching the Java code.
- **Reduced resource demand:** Lower-grade, hence cheaper, Java resources (developers) can be used. With SQLC Java developers don't need to use the JDBC API or any other tool-specific API, they use compiled methods, which, if named properly, are self-descriptive.



Pavel Vlasov is an IT architect with CitiCards in Jacksonville, FL. He has 11 years of IT experience and he's the author of Hamurapi, an open source code review tool. He can be reached at <http://www.pavelvlasov.com> or <http://www.hamurapi.org>.

vlasov@pavelvlasov.com

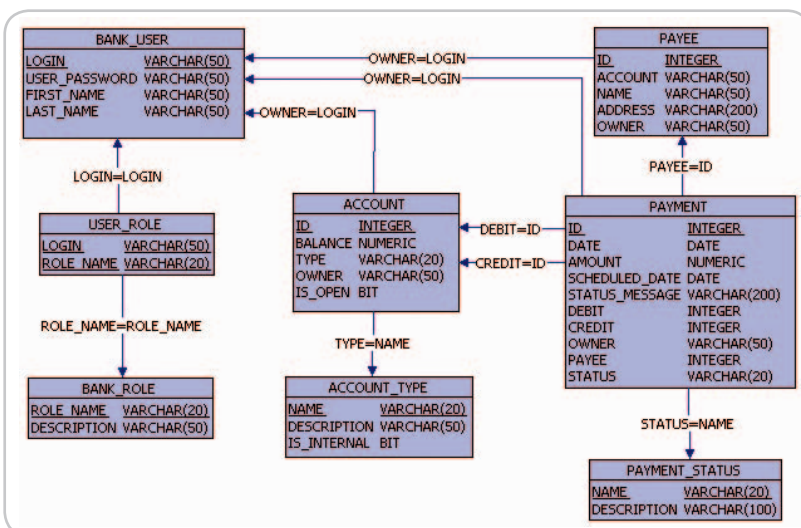


Figure 1

- **Increased productivity** according to Adam Smith.
- **Testability:** Statements can be tested independently of the Java application in the SQL console. Compiled classes can also be tested independently by, say, JUnit without complicated fixtures.

The following sections show how to use SQLC and then I'll compare SQLC, plain JDBC, and O/R mapping (using Hibernate as an example) applicability in different contexts.

Compilation

Figure 1 shows a sample model that will be used to generate classes and interfaces. We'll create a simple banking application capable of making fund transfers and collecting service charge from accounts with a low balance. The full source code of the sample application can be downloaded. See the references below.

The first step is to generate classes using an *sqlc* Ant task:

```
1.<sqlc
2.script="src/com/pavelvlasov/sqlc/samples/Bank.sql"
3.    dir="sqlc_generated"
4.    docDir="sqlcDoc"
5.    package="com.pavelvlasov.sqlc.samples"
6.    masterEngine="BankEngine"
7.>
8.    <table/>
9.</sqlc>
```

In the snippet above classes are generated using *Bank.sql* script. In this case a Hypersonic temporary database is created, the script is executed to create database objects, and then Java classes are generated. This approach works fine if the DDL in the script is compatible with Hypersonic. If you'd like to generate classes straight from the target database (the preferred method) then instead of the script attribute nested *<connection>* element shall be specified. This is a sample connection element for Oracle:

```
10.<sqlc
11.    dir="sqlc_generated"
12.    docDir="sqlcDoc"
13.    package="com.pavelvlasov.sqlc.samples"
14.    masterEngine="BankEngine"
15.>
16.    <connection
17.        driverClass="com.inet.ora.OraDriver"
18.        url="jdbc:inetora:server:port:DB"
19.        user="DBUSER"
20.        password="DBPASSWORD"/>
21.    <table/>
22.</sqlc>
```

Please note that it uses ORANXO driver because the drivers provided by Oracle itself are not fully JDBC-compatible (see the compatibility section below).

The SQLC task shown above produces .class files in the *sqlc_generated* directory and HTML documentation in the *sqlcDoc* directory using table metadata. The *<table>* element matches all the tables in the database.

Now we will use the generated *com.pavelvlasov.sql.samples.BankEngine* class to perform database operations. *BankEngine* constructor takes one argument of type *com.pavelvlasov.sql.SQLProcessor*. *SQLProcessor* can be created either from *java.sql.Connection* or *javax.sql.DataSource*. Our banking application uses *getProcessor()* and *getEngine()* helper method to obtain the engine:

```
23.private SQLProcessor processor;
```

```
24.
25.private SQLProcessor getProcessor() throws BankException {
26.    if (processor==null) {
27.        ...
28.    }
29.    return processor;
30.}
31.
32.private BankEngine engine;
33.
34.private BankEngine getEngine() throws BankException {
35.    if (engine==null) {
36.        engine=new BankEngine(getProcessor());
37.    }
38.    return engine;
39.}
```

In an application server environment the engine could be bound to the JNDI tree and obtained by the application through JNDI lookup.

We can't make (financial) transactions without creating customers, accounts, and other reference objects first. Below is a fragment of database initialization code that shows how to insert records in each of database tables except PAYMENT:

```
40.BankEngine engine=new BankEngine(processor);
41.engine.insertBankUser("joe", "pwd", "Joe", "Brown");
42....
43.engine.insertBankRole("customer", "Bank customer");
44....
45.engine.insertUserRole("joe", "customer");
```

We've got problems with your name on them.

At Google, we process the world's information and make it accessible to the world's population. As you might imagine, this task poses considerable challenges. Maybe you can help.

We're looking for experienced software engineers with superb design and implementation skills and expertise in the following areas:

- high-performance distributed systems
- operating systems
- data mining
- information retrieval
- machine learning
- and/or related areas

If you have a proven track record based on cutting-edge research and/or large-scale systems development in these areas, we have brain-bursting projects with your name on them in Mountain View, Santa Monica, New York, Bangalore, Hyderabad, Zurich and Tokyo.

Ready for the challenge of a lifetime?

Visit us at <http://www.google.com/jdj> for information. EOE



```

46. ...
47. engine.insertAccountType("checking",
    "Checking account", false);
48. ...
49. engine.insertAccount(1, new
    BigDecimal(1000), "checking", "joe", true);
50. ...
51. engine.insertPaymentStatus("scheduled",
    "Scheduled to be processed in the future");
52. ...
53. engine.insertPayee(1, "123-456", "KCP&L",
    "Kansas", "joe");

```

Simple, huh? No SQL involved. Database objects (Connection, Statement, ResultSet) lifecycle management (open/close) is encapsulated in SQLProcessor.

SQLC takes column and parameter nullability into account. If there's a column that maps to a Java primitive type and it is nullable then the corresponding wrapper class is used in generated interfaces and methods. For example, *Payment.getPayee()* returns *java.lang.Integer*. On the other hand, if the column/parameter isn't nullable then a primitive type is used. For example, *Payment.getId()* returns *int*.

Now let's get down to business and create a method that transfers funds from one account to another. We want the database to calculate the new account balance so we add two lines to *sqlc* task:

```

54.<update>name="AccountCredit">UPDATE ACCOUNT
SET BALANCE=BALANCE+? WHERE ID=?</update>
55.<update>name="AccountDebit">UPDATE ACCOUNT
SET BALANCE=BALANCE-? WHERE ID=?</update>

```

Then we run the task again. This results in two additional methods in *BankEngine*: *accountCredit(BigDecimal, int)* and *accountDebit(BigDecimal, int)*. We could also store the statements in a database table and use *<dbstatements>* nested element in the *<sqlc>* task or store them to an xml file and use a *<statements>* nested element.

Here is the *transfer()* method per se:

```

56. void transfer(BigDecimal amount,
    int debit, int credit, String owner,
    Integer payee) throws BankException,
    SQLException {
57.    BankEngine engine=getEngine();
58.    Account da=engine.getAccount(debit);
59.    if (da.getBalance().
        compareTo(amount)<0) {
60.        throw new InsufficientFundsException(
            "There are not enough funds
            on account "

```

```

62.        +da.getId()
63.        +". Required: "
64.        +amount
65.        +", available: "
66.        +da.getBalance());
67.    }
68.    engine.accountDebit(amount, debit);
69.    engine.accountCredit(amount, credit);
70.    Date now=new Date(System.currentTimeMillis());
71.    engine.insertPayment(
72.        getProcessor().
        nextPK("PRIMARY_KEY",
        "PAYMENT"),
73.        now,
74.        amount,
75.        now,
76.        "Processed",
77.        debit,
78.        credit,
79.        owner,
80.        payee,
81.        "processed");
82.}

```

Now we'll take a look at how SQLC generates methods using index information. We need to collect service charges on accounts with a balance below some limit. So first we need to iterate over such accounts. It's not a bad idea to use an index on the BALANCE column to make iteration more effective. We'll name index IX_ACCOUNT_SQLC\$_M_BALANCE. SQLC\$_ to tell SQLC to generate methods using this index. M is the index mode. BALANCE is the postfix for the generated method. Using this information SQLC generates *getAccountBalanceLE()* methods that we'll use in our first implementation of *serviceCharge()* method:

```

83.void serviceCharge(BigDecimal limit,
    BigDecimal charge, int chargeAccount) throws
    BankException, SQLException {
84.    BankEngine engine=getEngine();
85.    Iterator it=engine.getAccountBalanceLE(
        limit).iterator();
86.    while (it.hasNext()) {
87.        Account account=(Account)
            it.next();
88.        if (!account.getOwner().equals(
            "_bank") && account.
            getId()!=chargeAccount && account.
            getBalance().compareTo(ZERO)>0) {
89.            BigDecimal amount=charge.
                min(account.getBalance());
90.            transfer(amount, account.
                getId(), chargeAccount,
                "_bank", null);
91.        }

```

```

92.    }
93.}

```

While the method above demonstrates the use of the index-generated method it has several flaws:

- All account types are charged in the same manner
- We had to use if-condition to avoid charging accounts that belong to the bank itself or accounts with no balance.

To improve the method we'll add a query to *sqlc* task:

```

94.<query>name="AccountSubjectToServiceCharge">
95.SELECT * FROM ACCOUNT WHERE BALANCE >? 0
96.AND BALANCE <? ? AND TYPE=?</query>

```

This will result in the generation of a *getAccountSubjectToService-Charge()* method that will be used for enhanced service charge collection:

```

97. void serviceChargeEx(String account
    Type, BigDecimal limit, BigDecimal charge,
    int chargeAccount) throws BankException,
    SQLException {
98.    BankEngine engine=getEngine();
99.    Iterator it=engine.getAccountSubject
        ToServiceCharge(limit, account-
        Type).iterator();
100.    while (it.hasNext()) {
101.        Account account=(Account)
            it.next();
102.        BigDecimal amount=charge.
            min(account.getBalance());
103.        transfer(amount, account.getId(),
            chargeAccount, "_bank", null);
104.    }
105.}

```

And finally the main method of the application:

```

106. public static void main(String[] args)
    throws Exception {
107.    Bank bank=new Bank();
108.    bank.transfer(new BigDecimal(250), 1,
        1000000, "joe", new Integer(1));
109.    bank.serviceCharge(new
        BigDecimal(500), new BigDecimal(5),
        1000001);
110.    bank.serviceChargeEx("checking", new
        BigDecimal(500), new BigDecimal(5),
        1000001);
111.    bank.serviceChargeEx("savings",
        new BigDecimal(5000), new
        BigDecimal(15), 1000001);
112.}

```

The following sections summarize SQLC capabilities.

Tables

From a table SQLC generates:

- Interface with accessors and (optionally) mutators for all table columns.
- Implementation of the interface (Java Bean).
- Select methods that return all table rows.
- Select method returns a single row by a primary key (if the table has primary key).
- Delete method, which deletes all rows.
- Delete method deleting a row by a primary key (if the table has primary key).
- Insert method with as many parameters as there are columns in the database.
- Insert method with one parameter of the type of the generated table's interface.
- Update method with one parameter of the type of the generated table's interface. The method is generated if the table has both primary-key and non-primary key columns. The method updates non-primary key columns using primary-key columns in the WHERE clause.

Queries (Select Statements)

For each query SQLC generates:

- Interface with accessors and (optionally) mutators for all query columns. SQLC tries to reuse/extend already defined interfaces.
- Implementation of the interface.
- Select methods, which return database-backed collection. The primary purpose of database-backed collection is to iterate over results without fetching them all to memory.
- Select method, which puts query results in provided collection.

Updates (Insert, Update, Delete Statements)

For DML statements SQLC generates one method per statement.

Indices

For an index, whose name matches the generation-policy template, SQLC generates, depending on the mode(s) specified:

- Select for all rows ordered by index.
- Select and delete rows with equal index columns
- Select and delete rows with non-equal index columns
- Select and delete rows with positions in the index less than those specified in the parameters
- Select and delete rows with positions in the index less or equal than those specified in the parameters
- Select and delete rows with positions in the index greater than those specified in the parameters
- Select and delete rows with positions in the index greater or equal to those specified in the parameters.

The reason behind generating methods from indices is that the generated method uses the index to do the operation. This is a mechanism to secure the application from long-running queries running on unindexed columns.

Automatic Interface Inheritance

SQLC automatically builds a hierarchy of generated interfaces. For example, the *AccountType* interface extends the *PaymentStatus* interface because both have *getDescription()* and *getName()* methods. It may seem insane at first time but don't forget – the interfaces just represent data and don't bear any other semantics. Inheritance is convenient for writing generic data-processing algorithms.

It's also possible to get SQLC to reuse existing interfaces or generate

interfaces that extend existing interfaces. It's done by adding an *<interface>* element to the *<sqlc>* task.

One more feature in interface generation is the generation of “common denominator” interfaces. A common denominator interface is generated if there are two or more interfaces that start with the same word(s) and have common methods. For example, if there are two queries that generate two interfaces – *PersonUsa* and *PersonRussia*:

```
113. public interface PersonUsa {
114.     int getId();
115.     String getLastName();
116.     String getFirstName();
117.     String getSsn();
118. }
119.
120. public interface PersonRussia {
121.     int getId();
122.     String getLastName();
123.     String getFirstName();
124.     String getPassportNo();
125. }
```

SQLC will find that:

1. Both queries names start with the word 'Person.' A word is defined as a sequence of characters starting with a capital letter.
2. Queries have common columns. And it will generate the interface *Person*. *PersonUsa* and *PersonRussia* will extend the interface *Person*:



The advertisement for Common Controls features a blue header with the company logo and website. Below, it promotes 'The Java™ Presentation framework for J2EE™ Web applications' based on Java™, Servlets™, Java Serverpages™, and Struts. A 'For Free!' badge is prominently displayed. Text encourages users to get a free trial version from the website and see common controls in action through an online demo. It lists various control elements like Lists, Trees, Tabfolders, Menu, Forms, BreadCrumbs, Calendar, and Colorpicker. The footer repeats the website URL.

```

126. public interface Person {
127.     int getId();
128.     String getLastName();
129.     String getFirstName();
130. }
131.
132. public interface PersonUsa extends Person {
133.     String getSsn();
134. }
135.
136. public interface PersonRussia extends
    Person{
137.     String getPassportNo();
138. }

```

Interface Implementations

Generated-interface implementations are serializable and implement equals(), hashCode(), toString(), clone(), and toDom(Element) methods.

There are two types of implementations – “dumb” for queries and “smart” for tables. Smart implementations keep track of changes and when insert or update is executed then an SQL clause is constructed on-the-fly and contains only modified columns for inserts and modified columns plus primary-key columns for updates.

Dealing with Big Databases

If you have a database with hundreds of tables it's not a good idea to generate classes for all of them in one package and one engine. You should compartmentalize your schema into manageable subject areas and generate classes for each area in a different package.

Instead of the `<table/>` element you'll need to explicitly provide catalog, schema, and table names. For example, to generate classes for all tables in the BANK schema you'll need to add the `<table schema="BANK"/>` to the *sqlc* task.

Database Standards and Generation Policy

SQLC uses the names of database objects to generate Java classes and methods. With this database standards gain clear purpose – table and field names will comply with an organization's naming conventions not just because pesky DBAs want to show their power, but because it's needed to generate an intuitive and comprehensible Java API from the database.

Different organizations have different DB standards. And a standard isn't something that can be changed easily. For example, according to DB standards at

the organization where I worked before the field BALANCE in the ACCOUNT table should have the name ACCOUNT_BALANCE_NBR.

That's OK, but I don't want to have a Java property *AccountBalanceNbr*, as SQLC would generate by default. I want it to be Balance. The solution is to provide a custom implementation *com.pavelvlasov.sql.metadata.GenerationPolicy*. The easiest way to do that is to subclass *com.pavelvlasov.sql.metadata.DefaultGenerationPolicy*. To solve the problem a *generateColumnName()* method should be overridden.

Transactions

SQLC doesn't mess with transaction management. Transactions are attributes of the invocation context. For example, in EJB when you obtain a datasource and then a connection inside a method that has transactional attribute set that connection should already be properly enrolled in a transaction.

SQL Execution Metrics

One more buy-in for SQLC, dear reader. Compiled classes use a *com.pavelvlasov.sql.SQLProcessor* class for all database requests. An SQLProcessor class can gather JDBC statistics for you – the SQL statement that was executed, how many times, min, max, average and total execution time. You can read about metrics framework on www.pavelvlasov.com.

Database and JDK Compatibility

SQLC requires JDK 1.4 and compatible JDBC drivers. Drivers should implement *Connection.getMetaData()*, *PreparedStatement.getMetaData()* and *PreparedStatement.getParameterMetaData()*.

This is a list of databases that I tested for SQLC compatibility.

Compatible:

- Oracle 9.2 with the ORANXO driver
- Hypersonic 1.7.2
- Firebird 5.1
- Cloudscape 10

Incompatible:

- MySQL 4.2 – According to MySQL's JDBC driver documentation, there is no such paradigm as a statement parameter in the MySQL engine so parameterized statements are 'emulated' and the implementation of *getParameterMetaData()* requires a parser embedded in the MySQL JDBC driver.

- Oracle 9.2 with Oracle drivers – Oracle drivers throw an exception with the message “Unsupported feature” when the *getParameterMetadata()* method is invoked.
- Sun ODBC-JDBC bridge from JDK 1.4 (tested with Oracle 9.2)

Please note that incompatibility doesn't mean that you can't take advantage of SQLC when you use an incompatible database. If your statements and database DDL are compatible with Hypersonic you can use the script element or attribute of the *sqlc* task to have Hypersonic provide the metadata information needed and then use compiled classes with your target database. I did it with Oracle – it worked just fine.

Compiled classes don't use any JDK-1.4 features and should be 1.3.x-compatible, but I didn't test them on Java 1.3.

Aerobatics

In this section I'll describe several advanced features.

The first one is changing the visibility of the generated classes and method. By default, all classes and methods are public. This can be changed to, say, package visibility, by setting the *engineVisibility*, *engineMethodsVisibility* and/or *interfaceMethodImplVisibility* attributes of the SQLC task.

In previous sections you saw an example of the projection: *getAccountSubjectToServiceCharge(BigDecimal, String)* method. It returns a collection of *AccountImpl* objects. It is possible to have *getAccountSubjectToServiceCharge()* return a collection containing elements of another type. If you take a look at the BankEngine class you'll find that there are several *getAccountSubjectToServiceCharge()* methods with different signatures. Those methods that have *java.lang.Class targetClass* as a last parameter do the following trick: if the targetClass is a subclass of the automatically generated implementation then targetClass is instantiated and populated from the database row. Otherwise the targetClass shall have a single-argument constructor of a type compatible with the automatically generated implementation. In the latter case the automatically generated implementation (*AccountImpl*) is instantiated, populated, and then passed to targetClass constructor.

There are also methods that have Converter as a last parameter. Such methods ask Converter to convert an instance of, say, *AccountImpl* to something else and

then put that something else into a collection to be returned. The code snippet below is taken from Jsel. It demonstrates the use of conversion:

```
1. getSqlEngine().getExternalSupplierClients(  
140.     getName(),  
141.     getScanNumber(),  
142.     packageName,  
143.     new Converter() {  
144.         public Object convert(Object source) {  
145.             return getCompilationUnit(((Number) source).intValue());  
146.         }  
147.     }  
148. });
```

The last trick is the injection of SQLProcessor into instantiated collection elements. This is very straightforward – if the object to be put into collection implements *com.pavelvlasov.sql.DataAccessObject* then the SQLProcessor does the projection will inject itself into the instance.

PaymentImplEx class in the sample application and the *HistoryInspector*. *JoinedHistoryImplEx* class in Hammurapi (<http://www.hammurapi.org>) demonstrate the use of the advanced features.

Hibernate or Incarnate

This is a philosophical section that ruminates on what is primary – data or Java objects? If data is primary then hibernating Java objects is wrong – what we should do is incarnate data in a convenient way in Java. SQLC does just that.

So what's the answer? The answer is that there is no single answer it all depends on the situation. If you already have existing Java classes that you need to store in a relational database and the amount of data you need to store isn't too big then O/R mapping works very well.

If the amount of data is huge then the database modeling would, probably, need special attention and thus data becomes primary.

Data also is primary in a RUP-like development process. Illustration 6 elucidates the point. The process goes in the direction of lowering the level of abstraction. In other words, each step adds more detail.

The process starts by identifying requirements. After that Use Case and Business Object Domain (BDOM) models are produced. The Use Case model represents system behavior; BDOM represents system data - subject area entities and their relationships. BDOM objects have attributes but don't have operations. So, in fact, BDOM is a logical model of a database from which a physical model is produced by adding a few details – attribute types and constraints.

The Java class model, on the other hand, is a convergence of the Use Case model and BDOM. It requires the addition of more detail and so is a lower level of abstraction than the database model. The fact that RDMS semantics can be expressed in the Java language (pure Java databases like Hypersonic are an example), but not vice versa is another justification for Java having a lower level of abstraction than SQL.

Therefore, producing a database schema from Java classes is conceptually wrong because it violates the principle of lowering the level of abstraction.

Handcrafting Java classes from the database model is also a bad idea. Just duplicating database semantics is a waste of time, since it can be done automatically. Mixing business semantics with persistence semantics is even worse.

This is where the SQLC idea comes from – the database should be developed first. Then bridge classes should be generated by SQLC and only after that should the database-dealing Java classes be developed.

You may say – wait a minute, when the SQLC-generated classes incarnate data they produce objects, so SQLC is essentially an O/R mapping tool! No, it's not a mapping tool; it's a bridge-generating tool. Mapping assumes that

each table and column table name should be mapped to the name of an already existing class and the column name should be mapped to a Java property or field of that class. Typically such a mapping is done in XML files. Another point proving that SQLC is not an O/R mapping tool is that the object is data+behavior. Objects produced by SQLC-generated classes and representing rows don't have business behavior – just data accessors/mutators. So they are essentially data structures.

After reading this section one reviewer remarked: "Who believes in RUP?" I think some people do, but that's not the point. The point is that if your application has just a dozen business-domain objects and your developers are capable of envisaging and coding them right off the bat in Java then the O/R mapping path is the right one. But if, for whatever reason, you tend to create the database model first then SQLC is probably a better option.

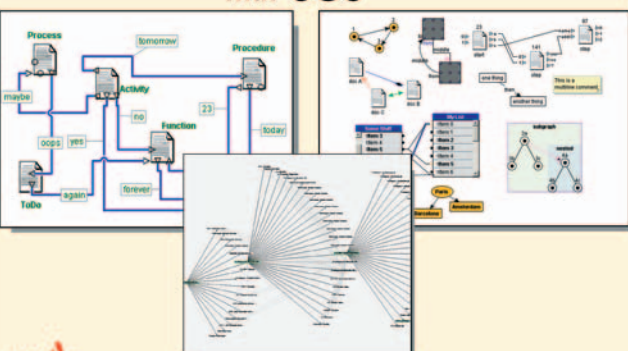
Applicability Comparison

By now you've got an idea what SQLC is and what it does. As I said SQLC isn't an O/R mapping tool. It doesn't hibernate Java objects to a relational database; it incarnates database objects in Java. As such it can't be compared with, say Hibernate, like an M-16 can't be compared with a Hatori Hanzo sword. Both of them use the same basic principle, serve the same purpose, but do it differently. What can be compared is their applicability depending on the context and the person wielding the tool.

JDBC

The services offered by JDBC are too low-level. Working with a database through plain JDBC is like writing an XML file using OutputStream – it's possible but the solution is poorly maintainable and error-prone.

Build Incredible Interactive Diagrams with JGo™




New!
JGo for SWT/Eclipse
JGo Instruments for meters, dials, gauges

Create custom interactive diagrams, network editors, workflows, flowcharts, and design tools. For web servers or local applications. Designed to be easy to use and very extensible.

- **Fully functional evaluation kit**
- **No runtime fees**
- **Full source code**
- **Excellent support**

Free evaluation at:
www.nwoods.com/go
800-434-9820 or 603-886-9173



With plain JDBC a developer can mistype table or column name, forget to close JDBC resources, issue queries that run for hours because the proper index wasn't created, and do a lot of other scary stuff.

Using plain JDBC also implies the developer knows SQL and leads to a mix of Java and SQL.

JDBC is the most flexible. All other tools are built on it, but at the same time working with plain JDBC is the most code-intensive approach.

Hibernate

Hibernate, no doubt, is a great O/R mapping tool. It also has many features that SQLC doesn't address, for example, caching and relationship navigations. Preferring it to SQLC depends on whether your developers are already familiar with Hibernate.

If they're not or if you're going to outsource development then Hibernate can be costly. Hibernate is open source and doesn't cost a groat, but the Hibernate reference manual is 140 pages long and Hibernate book is 400 pages. Effectiveness in Hibernate requires knowing tools like Xdoclet and Middlegen.

In the case of in-house developers, the cost will take the form of low productivity and a high level of rework during the learning period. In the case of outsourcing, the cost will take the form of the higher rate of Hibernate-proficient contractors.

Also note that part of the cost may be for nothing. For example, specifics of your application may make the Hibernate cache not only useless but dangerous.

Caching works fine if your application is the only one modifying the database. In such a situation roundtripping to the database is required only on data modification (insert/update/delete). This scenario corresponds to the commit option A for EJBs.

If there's more than one process updating the database at the same time then the only advantage of caching is reusing the instances (commit option B in EJB).

But in critical applications reusing instances may not be acceptable to avoid any chance of manipulating with stale data (commit option C in EJB). This is where caching shouldn't be used at all.

Transparent relationships management also isn't as good as it appears. Misunderstanding the concept can lead to a mess. Trust me, I've seen it.

Reportedly roughly 80% of a typical software system cost occurs *after* initial de-

ployment. A development team is typically gone shortly after initial deployment. So skillset requirements apply to maintainers too.

iBatis SQL Maps

This is another Java-DB tool. iBatis doesn't generate Java classes and methods. It maps a database to existing Java classes. So it's more about flexibility through indirection (mapping). A developer needs to handcraft Java classes and XML maps. An iBatis Java API takes the map name as a parameter. Problems like typos in the map name or in the map's SQL, or SQL being out of sync with the database can't be revealed at build time, only at runtime.

SQLC

SQLC is best applied in environments where:

- Databases are created according to database naming standards that generate self-descriptive names for Java methods and interfaces.
- Database objects are created before the Java code dealing with them is written.
- DBAs and data modelers are more permanent than developers.
- Development teams are volatile – one team works on phase one, another on phase two and another does support.
- Gartner says only 32% of the 2.5 million Java developers in the world really know Java, which means there's a serious shortage of high-level development skills. SQLC benefits organizations that employ not only the smart third but also those who don't possess genuine Java knowledge.
- The Java application being developed doesn't need to run on a dozen different RDBMS with different table and column names. In other words configurability isn't the key. SQLC and SQLProcessor support SQL templating through `${property}` substitution in SQL text at runtime. But this functionality is very rarely needed.

SQLC doesn't cover all data-access scenarios and there will be times when you'll need to fall back to Squirrel or plain JDBC.

Conclusion

You'd probably expect to find something like this in the conclusion: *"It's not possible to describe all the cool features of product X*

in a magazine article. There's an excellent book (written by the author ..." Not in this article. What you just read is pretty much the complete SQLC user manual.

Probably the best way to find out whether SQLC is right for you is to give it a try. Download a sample application and play with it. After that you'll make your choice – to use O/R mapping tools, plain JDBC, SQLC, or a mix of them. My previous employer did just this and saved 20% on the first project where SQLC was used. ☛

Resources

- Sample code for this article <http://www.pavelvasov.com/articles/sqlc/sqlc-samples.zip>
- SQLC: Manual – <http://www.pavelvasov.com/pv/content/Articles/sqlc/sqlc.html>; Presentation – www.pavelvasov.com/products/common/doc/SQLC.pdf
- Squirrel (<http://www.pavelvasov.com/pv/content/Articles/articles.sql.html>, Download - *Pvcommons*: www.pavelvasov.com/pv/content/downloads/index.jsp.html - classes simplifying common database operations and encapsulating JDBC resource management patterns.
- Antlr (<http://wwwantlr.org>) – Parser generator. Used by code generation classes.
- BCEL (<http://jakarta.apache.org/bcel/>) – Bytecode generation library.
- Hypersonic (<http://sourceforge.net/projects/hsqldb/>) – 100% Java SQL database.
- Ant (<http://ant.apache.org/>) – Java based build tool.
- Azzurri Clay (<http://www.azzurri.jp/en/software/clay/index.jsp>) – Database modeling plug-in for Eclipse.
- Hibernate (<http://www.hibernate.org>) popular Java O/R mapping framework.
- *Software Architecture in practice*. Len Bass, Paul Clemens, Rick Kazman. ISBN 0-321-15495-9
- ORANXO (http://www.inetsoftware.de/English/Produkte/ORANXO/default_main.htm) – Mostly JDBC compatible fast driver for Oracle.
- iBatis SQL Maps (<http://www.ibatis.com/common/sqlmaps.html>) – Java – DB mapper.
- Jad (<http://kpdus.tripod.com/jad.html>)
- Jadclipse (<http://sourceforge.net/projects/jadclipse/>) will allow you to look inside generated classes if you want to.
- DJ Java Compiler (<http://members.fortunecity.com/neshkov/dj.html>) – nice GUI Java decompiler. Uses Jad behind the scenes.

Style Report 7

Light Weight, Integration Ready Enterprise Reporting & Analysis



open standards open architecture Linux Windows Java J2EE Tomcat SOAP LDAP DHTML

Traditional BI Challenges:

- ▶▶ Monolithic, resource intensive
- ▶▶ Proprietary software stack
- ▶▶ Requires ETL/Data warehouse

Style Report Solutions:

- ▶▶ Light weight, integration ready
- ▶▶ Open software stack, J2EE drop-in
- ▶▶ Real time transactional DB and OLAP



For more information and to download a free evaluation copy www.inetsoft.com/jdj



Know Your Worst Friend, the Garbage Collector

It can make or break performance



by Romain Guy

All Java programmers are aware of a peculiar entity living in their Java Virtual Machine known as the Garbage Collector. Although we all use it every day, only a few of us know exactly what it is and how it works. Unquestionably useful, the Garbage Collector can hurt the performance of your application without you even knowing it. In this article you will learn about its inner workings and understand how to tame it to boost your programs.

The Garbage Collector, which I will call GC from now on, is a benevolent sentinel present in every JVM. Its role is to identify and free chunks of memory left unused by the currently running application. Its job should shed some light on the origin of the name “garbage collector.” Despite the depreciating name, the GC is like Dilbert’s garbage man who is very smart, even smarter than you sometimes.

During its lifecycle, an application creates a certain number of objects, that is to say a certain amount of data that consumes memory and lasts according to its role in the inner workings of the application. Let’s take the simple example of a Web browser. The object corresponding to the window you look at has exactly the same life span as the application itself, while the object standing for the Google logo on the google.com front page lasts only as long as you’re on the page. As a matter of fact, an application spawns a large amount of short-lived objects during its life span. So you can understand that defining and controlling the lifecycle of every single object generated by an application demands a tremendous amount of work from developers.

A simple example should give you a better idea of the incredible number of objects that are born and then killed. When you open a plain text file in a text editor, in our case Jext, 342,997 objects are created and destroyed. Even the simple conversion of a pound to kilograms entails the birth and the death of more than 171,000 objects in Numerical Chameleon. Imagine if the programmer had to know exactly how to handle every single object during an application’s lifecycle. Yet, the

slightest error can prove disastrous; this is how infamous memory leaks happen. A memory leak is caused by objects, called undeads or zombies, left unused but still marked alive in memory. The more undeads wandering about, the more memory the application will need. The program eventually runs out of fresh memory and crashes. Good memory management is one of the most difficult issues in low-level languages like C or C++. Some high-level languages, like Java or Python, rely on a GC that cuts down the programmer’s workload. So a developer supported by a GC only needs to do object creation.

Although extremely convenient, a GC is not a miracle tool and every so often does wicked things. Sheltered behind this powerful shield against memory management, the developer can easily provoke a disaster and wind up blaming the GC, the language, or the platform for his own mistakes. By studying and understanding the inner workings of the GC you can optimize the Java Virtual Machine according to the specific needs of your applications. Best of all, you won’t have to change the source code. And you’ll be able to use these tricks with applications you didn’t write.

Garbage collectors have been around for a long time and dozens of algorithms have been devised to implement different collection strategies. The behaviors and properties we’re about to discuss are specific to Sun’s HotSpot JVM 1.4.1 and higher. Developers are free to choose their own memory reclaiming strategy when writing a GC, which is why we must focus on a particular implementation to learn how to optimize our programs accurately. You should also know that the same JVM version yields different results when executed on different platforms. For instance, the performance of Java software running on Solaris is boosted by tweaking the threads management.

The GC in the HotSpot JVM is also called the “generational garbage collector.” As its name suggests, this GC can make a difference to several generations of objects. Within the virtual machine, objects are born, live, and die in a memory area known as the heap. The heap itself is divided into two parts, each one corresponding to a given generation: the young space and the tenured space. The first hosts recent objects, also called children, while the second one holds objects with a long life span, also called ancestors. Next to the heap, the virtual machine contains another particular memory area, called the perm, in which the binary code of

Romain Guy, a long-time Java developer, is the author of the Jext source editor, free software available at jext.org. He is involved in other Open Source or free projects. Romain also works as a journalist for a French computing magazine and as a translator for O’Reilly France.

romain.guy@jext.org

each class loaded by the currently executing program is archived. Although the perm is important to applications dynamically generating a lot of bytecode, like a J2EE server, it's unlikely you'll ever need to tweak it. Beware though; some unexpected applications might need a lot of perm space. For instance, IBM's XSLT library Xalan consumes perm space when style sheets are memory-compiled. If you ever encounter a vicious `OutOfMemoryError`, think of the perm space. Figure 1 depicts the heap and its structure.

Both the tenured space and the young space contain a virtual space, a zone of memory available to the JVM but free of any data. That means that those spaces might grow and shrink with time. The way these spaces change their size is an important HotSpot setting. You can see three other memory areas in the young space in Figure 1: the eden and two survivor spaces. To understand their purpose we need to study the various algorithms used by the GC.

Whenever a new object is allocated to the heap, the JVM puts it in the eden. Survivors are treated in turn. The GC uses the one that remains free as a temporary storage bucket. When the young space gets overcrowded, a minor collection is done. A very simple copy algorithm is used that involves the free survivor space. During a minor collection, the GC runs through every object in both the eden and the occupied survivor space to determine which ones are still alive, in other words which still have external references to themselves. Each one of them will then be copied into the empty survivor space.

Quickness is the main advantage of this algorithm since it doesn't need to reclaim memory, strictly speaking. This copy algorithm is fostered by a feature in the modern JVM that ensures that the heap is a single contiguous memory segment. This is why it can't exceed about 1.5GB on Windows. Microsoft's OS splits the memory address space of a 32-bit process in half by allocating 2GB to the kernel and 2GB to the application. Theoretically the JVM should be able to allocate up to 2GB of contiguous memory but it can't because of the memory overhead used by the OS and the JVM itself. The perm is an example of this overhead. Anyway, at the end of a minor collection, both the eden and the explored survivor space are considered empty. As minor collections are performed, living objects proceed from one survivor space to the other. As an object reaches a given age, dynamically defined at runtime by HotSpot, or as the survivor space gets too small, a copy is made to the tenured space. Yet, most objects are born and die right in the young space. To obtain ideal performance, the JVM should be tweaked to avoid as many copies from the young space to the tenured space as possible.

In the tenured space the laws are different. Whenever more memory is needed, a major collection is done with the help of the Mark-Sweep-Compact algorithm. Though it's not complex, it's greedier than the copy algorithm. The GC will run through all the objects in the heap, mark the candidates for memory reclaiming, and run through the heap again to compact remaining objects and avoid memory fragmentation. At the end of this cycle, all living objects exist side-by-side in the tenured space. The major collections responsible for most Java applications slow down from time to time. Unlike a minor collection, running a major collection stops the execution of the whole application. So a good optimization trick is to reduce the burden of the Mark-Sweep-Compact algorithm.

Given this information, you should now be able to use the following JVM command-line options sensibly. Each `<size>` tag can be replaced by a size measured in bytes (i.e., 32,765), kilobytes (i.e., 96k), megabytes (i.e., 32MB), or gigabytes (i.e., 2GB).

- `-Xms<size>` specifies the minimal size of the heap. This option is used to avoid frequent resizing of the heap when your application needs a lot of memory.

- `-Xmx<size>` specifies the maximum size of the heap. This option is used mainly by server-side applications that sometimes need several gigs of memory. So the heap is allowed to grow and shrink between the two values defines by the `-Xms` and `-Xmx` flags.
- `-XX:NewRatio=<a number>` specifies the size ratio between the tenured space and the young space. For instance, `-XX:NewRatio=2` would yield a 64MB tenured space and a 32MB young space, together a 96MB heap.
- `-XX:SurvivorRatio=<a number>` specifies the size ratio between the eden and one survivor space. With a ratio of 2 and a young space of 64MB, the eden will need 32MB of memory whereas each survivor space will use 16MB.

Choosing a value for each parameter is a difficult job that depends solely on the application you're attempting to tweak. An application creating large amounts of short-lived objects, like an HTTP server for instance, won't have the same needs as a largely static application, like a screen saver. Sun provides a helpful document about many HotSpot options called Java HotSpot VM Options. You should also read A Collection of JVM Options by Joseph Mockler who gathered a comprehensive list of Sun's JVM command-line flags.

Besides the heap, HotSpot can change the behavior of the GC itself. Although the algorithms don't differ much, you can choose among three different execution modes. The GC interrupts the application for minor and major collections. This is a troublesome behavior on multiprocessors where you lose a lot of CPU power every time the GC is used. You can lessen, if not avoid, the impact on performance by carefully picking the GC execution mode.

The first one is called incremental garbage collection. As its name suggests, this mode does a bit of major collecting every time a minor collection is run. Global performance suffers from this execution mode but you get rid of those nasty pauses caused by major collections. Incremental garbage collection also tends to leave the heap in a fragmented state. So it's a good idea to limit its use to applications with long-lived objects. It can help you in a subtle way to obtain more responsive Swing UIs.

The second mode is a parallel GC using many threads at once, at least one per processor, to do minor collections. If your target computer has less than three processors, it is unlikely you will ever notice a difference. This option is particularly well suited to heavy servers running applications spawning many short-lived objects like web servers.

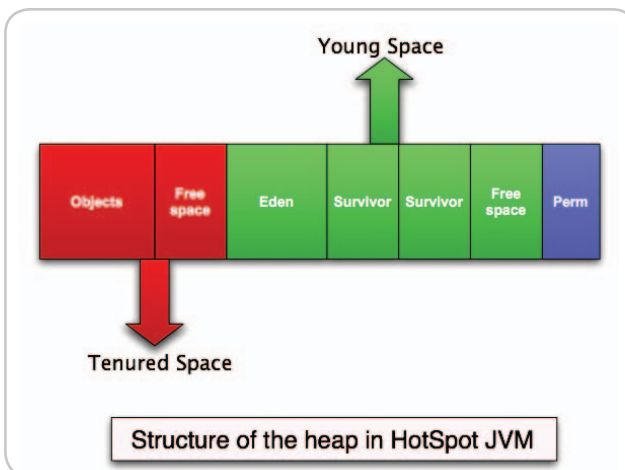


Figure 1 The heap is a contiguous memory area divided into two major spaces

Finally yet importantly, a concurrent GC can do incremental major collection without interrupting the application for long periods of time. A concurrent GC can also use parallel execution for minor collections. According to Sun Microsystems, this execution mode gets good results with interactive applications running on a single-processor computer. To choose among these three modes and set them up, you need to use the following command-line flags:

- `-Xincgc` activates incremental garbage collection.
- `-XX:+UseParallelGC` activates the parallel GC. The number of threads used to do minor collections is defined by the `-XX:ParallelGCThreads=<a number>` flag.
- `-XX:+UseConcMarkSweepGC` activates the concurrent GC. Parallel minor collections can be activated as well by using the `-XX:+UseParNewGC` flag.

To measure the impact of your choice on your application, you'll want to use the `-verbose:gc` command-line flag. You'll then get a GC activity trace on the standard output. Redirecting the standard output to a file and running the following Python script will let you draw very useful charts as shown in Figure 2.

This script can be used by specifying a file name as the first parameter or by piping it directly to the Java command.

```
#!/usr/bin/env python
# -*- coding: ISO-8859-1 -*-
```

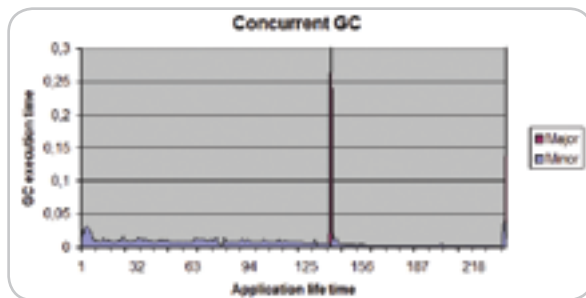


Figure 2 Major collections, in red, consume a lot of CPU power as opposed to minor collections, in blue

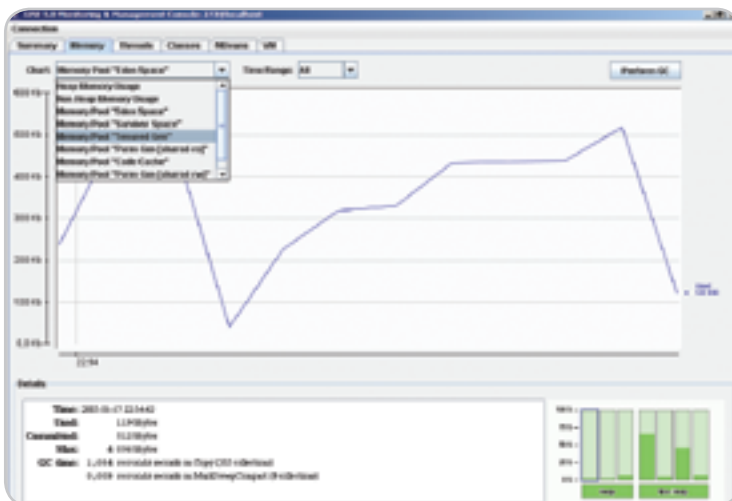


Figure 3 JConsole is a powerful and versatile tool to analyze memory consumption shipping with Sun's J2SE 5.0

```
import fileinput
import re

print "%s\t%s\t%s\t%s" % ("Minor", "Major", "Alive", "Freed")

for line in fileinput.input():
    match = re.match("(Full)?GC (\d+)K->(\d+)K((\d+)K\), ([^ ]+)"
        secs]", line)
    if match is not None:
        minor = match.group(1) == "Full " and "0.0" or match.group(5)
        major = match.group(1) == "Full " and match.group(5) or "0.0"
        alive = match.group(3)
        freed = int(match.group(2)) - int(alive)
        print "%s\t%s\t%s\t%s" % (minor, major, alive, freed)
```

The result is a CSV document using tabs as separators and containing four columns of data. The first one gives the time consumed by the GC to do a minor collection, the second one gives the same information for a major collection, the third one indicates the amount of memory used by the application after the execution of the GC and the last one gives the amount of memory reclaimed by the last collection. (This script was inspired by an AWK script written by Ken Gottry in the article *Pick up performance with generational garbage collection* published on java-world.com.)

The chart presents the running time of each collection performed by the GC during the application life cycle. This chart was generated with the following command line and OpenOffice.org:

```
$ java -verbose:gc -Xms64m -Xmx128m -XX:NewRatio=2 -
XX:+UseConcMarkSweepGC -jar ../lib/jext-5.0.jar > gclog
$ gclog2csv.py gclog > gclog.csv
```

Although simple, this small script can help you tune your application pretty quickly. As of J2SE 5.0, Sun Microsystems provides a more powerful tool called JConsole that will graphically monitor the state of the heap and the activity of the GC as shown in Figure 3.

Further Reading

- *Java HotSpot VM Options*, Sun Microsystems, <http://java.sun.com/docs/hotspot/VMOPTIONS.html>
- *A Collection of JVM Options*, Joseph Mocker, <http://blogs.sun.com/roller/resources/watt/jvm-options-list.html>
- *Pick up performance with generational garbage collection*, Ken Gottry, <http://www.javaworld.com/javaworld/jw-01-2002/jw-0111-hotspotgc-p4.html>
- *Garbage Collection in the Java HotSpot Virtual Machine*, Tony Printezis, http://www.devx.com/Java/Article/21977?trk=DXRSS_JAVA
- *Garbage Collection Performance*, Brian Goetz, <http://www-106.ibm.com/developerworks/java/library/j-jtp01274.html>
- *Using JConsole to Monitor Applications*, Mandy Chung, <http://java.sun.com/developer/technicalArticles/J2SE/jconsole.html>

Advertiser	URL	Phone	Page
Altova	www.altova.com	978-816-1600	4
Borland	www.borland.com/jbuilder	831-431-1000	7
Business Objects	www.businessobjects.com/dev/p26	888-333-6007	13
ceTe Software	www.dynamicpdf.com	800-631-5006	43
Common Controls	www.common-controls.com	+49 (0) 6151/13 6 31-0	53
DataDirect	www.datadirect.com/jdj	800-876-3101	Cover IV
Google	www.google.com/jdj	650-253-0000	51
ILOG	http://diagrammer.ilog.com	800-FOR-ILOG	15
InetSoft	www.inetsoft.com/jdj		57
Information Storage & Security Journal	www.issjournal.com	888-303-5282	61
Java Developer's Journal	www.jdj.sys-con.com	888-303-5282	45
JavaOne Conference	www.java.sun.com/javaone/sf	866-382-7151	49
Jinfonet Software	www.jinfonet.com/jd5	301-838-5560	31
M7	www.m7.com/d7.do	866-770-9770	33
Microsoft	microsoft.com/connectedsystems		19
Microsoft Visual Studio	www.msdn.microsoft.com/visual		Cover II
Northwoods Software Corp.	www.nwoods.com/go	800-434-9820	55
Parasoft Corporation	www.parasoft.com/jtest_IDJ	888-305-0041	9
Phoneomena	www.phoneomena.com	352-373-3966	41
ReportingEngines	www.reportingengines.com	888-884-8665	23
Software FX	www.softwarefx.com	800-392-4278	Cover III
WebAppCabaret	www.webappcabaret.com/jdj.jsp	+61 3 6226 6274	39

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

This index is provided as an additional service to our readers. The publisher does not assume any liability for errors or omissions.

Subscribe Today!

— INCLUDES —
FREE
DIGITAL EDITION!
(WITH PAID SUBSCRIPTION)
GET YOUR ACCESS CODE
INSTANTLY!



The major infosecurity issues of the day... identity theft, cyber-terrorism, encryption, perimeter defense, and more come to the forefront in ISSJ the storage and security magazine targeted at IT professionals, managers, and decision makers

SAVE 50% OFF!

(REGULAR NEWSSTAND PRICE)

Only \$39⁹⁹

ONE YEAR
12 ISSUES

www.ISSJournal.com
or 1-888-303-5282

Industry News

Sun Java Application Platform Suite Achieves Best-in-Class

(Santa Clara, CA) – Sun Microsystems, Inc., has announced the publication of a SPEC-jAppServer2004 benchmark result demonstrating the best price/performance in the application tier for its Sun Java System Application Server 8.1 Standard Edition. The Sun system achieved a score of 1201.44 jAppServer Operations Per Second (JOPS) @ Standard, establishing a world record price/performance advantage for the Sun Java System Application Server especially when running on the Solaris 10 operating system (OS) and the Sun Fire server product line.

www.sun.com

Dieselpoint Search Version 3.5 Released

(Chicago) – Dieselpoint, a provider of search software, has announced the next-generation version of Dieselpoint Search, which provides a comprehensive all-Java solution to the problem of finding information in documents, databases, and XML.

Dieselpoint Search provides a solution to a challenge faced by many organizations that struggle with making it easy for end users to find and discover the information they need. Whether the organization is deploying online product catalogs, document search, site search, intranet/enterprise search, knowledgebases, or embedded OEM search, the challenge remains the same: How do we make it easy for our users to find what they are looking for quickly and efficiently?

www.dieselpoint.com

Wily Technology Offers Breakthrough Web Application Browser Response Monitor

(Brisbane, CA) – Wily Technology, a provider of enterprise application management software, has announced the availability of the Wily Browser Response Time (BRT) Adapter for monitoring Web application response to end-user browser requests. Wily's BRT Adapter is the first performance monitor to provide full, component-level transaction visibility from the browser to back-end systems with no hardware, no synthetic transactions, no application code changes, no security risk, and with low overhead. The Wily BRT Adapter is integrated with Wily Introscope, an application monitoring and management tool, and is fully functional out of the box.

www.wilytech.com

Oracle Fusion Middleware Enhances Application Foundation for Enterprise Services and Business Applications

(Redwood Shores, CA) – As part of its commitment to help organizations integrate and manage enterprise complexity, Oracle has announced Oracle Fusion Middleware and a roadmap for certifying PeopleSoft and JD Edwards applications with this middleware offering.

Oracle Fusion Middleware is the newly created brand for Oracle's family of existing middleware products, which includes all of the products needed to integrate a number of business applications.

Oracle plans to certify PeopleSoft and JD Edwards applications with Oracle Fusion Middleware products in Q2 and Q3 calendar 2005, respectively. Customers can use Oracle Fusion Middleware to support their entire enterprise including Oracle and non-Oracle

business applications, custom applications developed by in-house IT staff and consultants, and the emerging array of standards-compliant enterprise services.

www.oracle.com

NuComm International Uses JReport to Boost Productivity

Jinfony Software, a provider of 100% J2EE embedded reporting solutions, has announced that NuComm International, a Canadian-based provider of customer relationship and contact center services, has embedded JReport to provide integrated and automated reporting functionality to its NuStar performance management platform. NuStar serves as a process management system for developing, launching, and managing both client and internal campaigns. This includes conducting automated surveys, monitoring and managing inbound and outbound calls, and integrating voice recognition, chat and e-mail services.

JReport drives customer service and program responsiveness by creating reports that are fully actionable; analysts and end users can sort, filter, and drill through reports in real-time to meet specific data presentation requirements.

www.jinfony.com

Quest in Leaders Quadrant in Magic Quadrant for J2EE Application Server Management

(Irvine, CA) – Quest Software, Inc., a provider of application, database, and Windows management solutions, has announced that it has been listed in the leaders quadrant in Gartner's Magic Quadrant for J2EE Application Server Management, 2005. According to Gartner, the vendors listed in the leaders quadrant possess a large, satisfied installed base and have a high degree of visibility within the market.

Gartner Magic Quadrants evaluate vendors on their completeness of vision and their ability to execute. Leaders offer scalable, robust, and intelligent applications, and they address evolving enterprise requirements in the form of J2EE application life-cycle management.

Quest Software products that provide J2EE Application Server Management include Quest Foglight, Quest Spotlight, Quest PerformaSure, and the Quest JProbe Suite, which make up the Quest Application Performance Management Suite for the J2EE platform.

www.quest.com

JadeLiquid Software Releases WebRenderer v3.0

(Tasmania, Australia) – JadeLiquid Software has released WebRenderer v3.0, a Java browser component. WebRenderer v3.0 includes many customer-driven improvements with increased stability, all new packaging, and improvements in rendering speed.

The WebRenderer Java browser component integrates into any Java application or applet and provides standards compliant rendering of Web content across multiple platforms. WebRenderer, a standards-compliant Java browser component, supports HTML 4.01, SSL, JavaScript, CSS 1 and 2, XSL, XSLT, XML, DOM, etc. WebRenderer allows for the creation of HTML/XML/Web-content centric Java client applications by providing client-side Web content rendering for J2EE and Web infrastructure driven back ends. A trial version is available at www.webrenderer.com.

JadeLiquid
Software



SoftwareFX

Zero To Chart

In Less Than An Hour!



9:04 am

To learn more about the Chart FX products visit www.softwarefx.com and decide which one is right for your platform and target. Then download the 30-day trial version or the Chart FX for Java Community Edition which is a FREE, non-expiring, full development version.

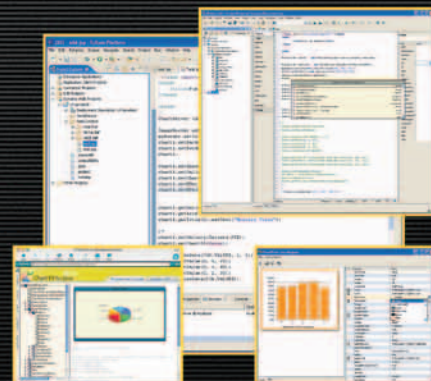
9:07 am

After download, simply install the product appropriate for your needs, platform and target environment.



9:13 am

Open the Chart FX for Java Designer to get started with the look and feel of your chart. Use your favorite IDE to add functionality and to populate the chart by connecting to your data source. The Chart FX Resource Center is also available to help with any questions you may have. Then deploy to your specific application server. ➤



9:58 am

Your application is then displayed with an active chart or image that makes any data look great!



Ready... Set... Download!


Chart FX

US: (800) 392-4278 • UK: +44 (0) 8700 272 200 • Check our website for a Reseller near you!

www.softwarefx.com

©2005 Software FX. All rights reserved. Chart FX is a registered trademarks of Software FX, Inc. All other brands are owned by their respective owners.

Remember the good old days?



Can you dig it? We miss shakin' our groove things, too. You can still get down to Nectarines and Spice's sweet and soulful sounds, but you wouldn't use an 8-track tape player to do it.

It's like that with Microsoft® SQL Server. Terrific database, but the JDBC driver available from Microsoft is old technology you wouldn't use today. You need better performance, reliability, and functionality. **DataDirect presents today's JDBC** – the SPECjAppServer/ECperf leader. Featuring Windows Authentication support, J2EE certification, and advanced 3.0 specification compliance for SQL Server 2000 and SQL Server 7.

Some things are better left in the past. Like that pale green leisure suit.

Why compromise on yesterday's technology?
Get current with **DataDirect Connect™ for JDBC**.

www.datadirect.com/jdj
(800) 876-3101


DataDirect™
TECHNOLOGIES

DataDirect Connect is a registered trademark of DataDirect Technologies. JDBC is a registered trademark of Sun Microsystems, Inc. in the United States and in other countries. DataDirect Technologies is independent of Sun Microsystems, Inc. Microsoft is a registered trademark of Microsoft Corporation.