

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

JDJ

MARCH 2005 VOLUME:10 ISSUE:3

IN THIS ISSUE...

FEATURES:

Beyond Patterns: Thinking Objects
PAGE 10

**Using JDBC & the Template Method
Pattern for Database Access**
PAGE 42

Casting Perlin's Movie Magic in Java3D
PAGE 56

LOG4J VS JAVA.UTIL.LOGGING



Which logging library is better for you?

PLUS...

▶ Development of
Component-Oriented
Web Interfaces

▶ Java Naming
Services Internals
▶ Java Annotation Facility-
A Primer

▶ Making
PDFs Portable
▶ SLOOH.com Delivers Astronomy
to the Mainstream

RETAILERS PLEASE DISPLAY
UNTIL MAY 31, 2005

\$5.99US \$6.99CAN

04>



You deploy to multiple platforms.
You need one solution.



InstallAnywhere

Powerful Multiplatform Deployment

- One Installer Project for Multiple Platforms and Multiple Languages
- Support for Windows, Linux, Solaris and more than 20 other platforms
- Optimized for complex enterprise deployment

■ Download today at ZeroG.com

ZERO G
SOFTWARE

XP: eXtremely Provocative?



Jeremy Geelan



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 Editors: **Seta Papazian**
 Online Editor: **Martin Wezdecki**
 Research Editor: **Bahadir Karuv, PhD**

Writers in This Issue

Calvin Austin, Matt BenDaniel, Jeremy Geelan, Mike Jacobs, Bill Kohl, Kishore Kumar, Ben Litchfield, Alex Maclinovsky, Joe McNamara, Kim Polese, Keith Reilly, Ajit Sagar, Krishan Viswanth, Joe Winchester, Alexey Yakubovich

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 Kristin Kuhnle, kristin@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.



In a world bristling with TLAs (Three-Letter Acronyms), it's interesting that one acronym that has often caused an upset in the world of software development should be one containing just two letters: XP. (No, not *that* XP. What we're talking about here is XP as in eXtreme Programming.)

Back in 2002, in a now-famous essay provocatively entitled "XP – That Dog Don't Hunt," an independent IT consultant called Bill Walton wrote: "My position is that XP, if it does not successfully address these fundamental problems, will fail of its own accord. ... First there were the ERP debacles of the 1990s. Then Y2K. Then Internet mania. Now XP says to the executive, 'The problem is, you've been going about this all wrong.'... The Indian, Russian, Chinese, and other outsourcing firms have been saying to these same execs, 'We understand what you want, and you can have it. And you can have it for 40 percent less.' Maybe XP is just what our foreign competitors have been waiting for."

Walton's essay sparked a huge controversy, and to this day you can use "XP is Evil" as a search string on Google.

Now the controversy has come to *JDJ*. In his article last month ("Why Use Extreme Programming?" [Vol. 10 issue 2]), Troy Holmes wrote that eXtreme Programming "was created by Kent Beck and Ward Cunningham back in 1996. XP was one of the first development processes that fell into the realm of iterative programming."

To which a highly indignant *JDJ* reader, Dennis de Champeaux, writing from San Jose, CA, has written to say this claim is "ridiculous."

"Even the first paper on the waterfall process by Royce in 1970 had feedback arrows – which were lovingly called 'salmon ladders' by others," de Champeaux points out. He adds: "An explicit iterative process was described by Boehm [Hans J. Boehm] before 1988. Thus the claim that Kent Beck created iterative thinking in 1996 is ridiculous."

de Champeaux's criticisms do not end there, and his objections seem to encompass XP itself, not merely Holmes's article about it. "Kent Beck keeps advocating that software development is mainly programming," de Champeaux thunders. "This is an activity in the solution space, while the key issue

remains to figure out what customers want," he continues. "Programming is totally terrible for that activity."

"One of the fundamental differences in the planning phase used in XP is the implementation of user stories as a way of capturing use cases," Holmes wrote in his article. So far as de Champeaux is concerned this is a totally unacceptable method: "Jacobson's use cases are part of early OO analysis activities and can be represented best in English or – if desired – in UML use case diagrams. It is a very bad idea to start programming when one has not yet even finished an initial dialog with the customer to figure out what the problem is."

The fundamental goals of XP, as summarized in Holmes's article, were "to increase communication, simplify the development process, and obtain feedback from the customer to ensure that requirements were met."

Dennis de Champeaux may be only one reader, but he claims to represent many: "This is a serious insult to those that have worked for decades to develop software development methods (no not methodologies)," he declares, his contempt for XP nowhere clearer than in the way he begins his letter of protest: "For many years we have to endure nonsense from Kent Beck on eXtreme Programming. Now we find a warm-over by Troy Holmes about it in *JDJ*. Why? Why?"

The answer is that we strive to cover software development – not even just Java – from as many different, and sometimes even opposing, perspectives as possible, safe in the knowledge that the *JDJ* readership worldwide is perfectly able to pick and choose from among the various methodologies (or methods, as de Champeaux insists), just as they are able to pick and choose for themselves from among the many tools, services, and solutions we discuss each month in these pages – and from among the hundreds of columnists and the thousands of writers that we have published over the past nine years.

We look forward to continuing that agnostic approach for the next nine years; meantime, keep that spirited feedback coming – including where you stand on eXtreme Programming. ☺

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



DESKTOP



CORE



ENTERPRISE



HOME

Bring your development plans to light

Sneak a peek at XMLSpy® 2005,
the industry leading XML development
environment from Altova®.

Revealed in Version 2005:

- XSLT 2.0 support and debugger
- XPath 2.0 support and analyzer
- XQuery 1.0 support and debugger
- Eclipse integration

See for yourself why XMLSpy 2005 is the standard for modeling, editing, debugging and transforming all XML related technologies. Illuminate your strategy with advanced standards compliance, innovative development and analysis tools, and extended platform integration. Use XMLSpy 2005 to structure XML Schemas and devise XML documents, then automatically generate runtime code in multiple programming languages. Become a markup mastermind!

Download XMLSpy® 2005
today: www.altova.com

JDJ contents

JDJ Cover Story

Log4j vs java.util.logging

Which logging library is better for you?

by Joe McNamara

46

FROM THE GROUP PUBLISHER

XP: eXtremely Provocative?

by Jeremy Geelan

.....3

VIEWPOINT

From Here to Ubiquity

by Kim Polese

.....6

JAVA ENTERPRISE VIEWPOINT

SOA, MSOA, and Java

by Yakov Fain

.....8

COMPARTMENTS

Development of Component-Oriented Web Interfaces

A case for the Vitrage Framework

by Alex Maclinovsky & Alexey Yacubovich

.....14

AP/S

Java Naming Services Internals

Implementing a simple client/server-based JNDI naming service

by Kishore Kumar

.....28

CORE AND INTERNALS VIEWPOINT

Ten Years of Java Technology

by Calvin Austin

.....34

TECHNIQUES

Java Annotation Facility – A Primer

JDK 5 has changed source code generation in a seminal way

by Krishan Viswanth

.....36

DESKTOP JAVA VIEWPOINT

Go Fast It Runs Too Slow

by Joe Winchester

.....50

DOCUMENTS

Making PDFs Portable

Integrating PDF and Java technology

by Ben Litchfield

.....52

@ THE BACKPAGE

SLOOH.com Delivers

Astronomy to the Mainstream

by Matt BenDaniel

.....62

Features

10

Beyond Patterns: Thinking Objects

by Bill Kohl

42

Using JDBC & the Template Method Pattern for Database Access

by Keith Reilly

56

Casting Perlin's Movie Magic in Java 3D

by Michael Jacobs



Kim Polese

From Here to Ubiquity

Ten years after we officially launched Java in May 1995, our dream of a ubiquitous software platform to power a networked world has actually come true. Today, some form of Java runs on 1.4 billion devices, and there are more than 4.5 million Java developers worldwide. Mobile applications like Java-based digital wallets generated more than \$1.4 billion for the almost 100 mobile carriers who use Java worldwide in 2003.

Java was, quite simply, the right technology at the right time.

In the early 1990s, Java's architects at Sun anticipated a world in which a ubiquitous public network would connect devices of all kinds and let people collaborate on an unprecedented scale. That was only a few years before the Mosaic browser was released and the Web was born.

When we planned the launch of Java (then called "Oak"), our goal was *ubiquity*. We knew we had a powerful technology, but our challenge was finding the right platform on which to launch it. Initial forays in the nascent PDA and interactive TV markets proved premature, but our persistence paid off when we downloaded an early version of the Mosaic browser.

We realized that the World Wide Web was the ideal platform to launch Java. Exciting as it was, Mosaic displayed only static text and images. What was missing was interactivity: the ability to run a program in the Web page, see animations, and get a real-time response. So the team developed HotJava, the world's first interactive browser, so people could see animations, live stock quotes, sports scores, and other data come alive on the Web.

From the day we released the newly renamed Java — along with HotJava, the full spec, and the source code — developers embraced it.

Technically, Java broke through platform barriers. It freed developers from proprietary hardware, and let them write applications once for many different operating systems. It was flexible. As a language, Java was designed to be small enough to run even on low-powered mobile devices, but complete enough to support complex applications. And by using a virtual machine, Java could address security problems that had foiled previous

attempts to create portable code.

In assessing Java's business potential, Sun's top executives realized the potential to encourage widespread adoption of this powerful technology through free distribution combined with innovative licensing terms. With the support of Eric Schmidt and Bill Joy, we put the full spec and source code for Java online, while Sun retained the licensing rights.

We were convinced that freely distributing the system to individual developers was the only viable path to ubiquity. Java was made freely available for download, which spurred thousands of software developers to build "applets," fueling Java's growth and adoption by showing off the potential of the Web.

When thousands of companies, from startups to major telcos and consumer electronics manufacturers, adopted Java to deploy new network-based services, its success was ensured.

Today, innovation in software is coming from another powerful phenomenon: open source development.

Java benefited greatly from shared learning and the collaborative development of hundreds of thousands of software developers. As an early stepping-stone in the new era of software design, Java showed what global, dynamic collaboration between individual developers could do. The current open source phenomenon shows the success of that approach: speedier deployments, dramatic cost savings, and often more reliable software systems.

The parallels are clear. In fact, Java's success derives from principles that are central to the growth of open source software:

First, the key to ubiquity is to make a technology *freely available*. Profits come from elsewhere: the value-added around the technology. Companies like MySQL and Red Hat have validated this model.

Second, technologies that allow *greater independence from proprietary standards* win. Java was an important step in liberating developers from proprietary hardware. Now, open source technologies are freeing enterprise IT from dependence on proprietary software.



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 Kuhle 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 Editors:
Seta Papazian seta@sys-con.com
Online Editor:
Martin Wezdecki martin@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
Richard Silverberg richards@sys-con.com
Assistant Art Director:
Andrea Boden andrea@sys-con.com

Web Services

Information Systems Consultant:
Robert Diamond robert@sys-con.com
Web Designers:
Stephen Kilmurray stephen@sys-con.com
Matthew Pollotta matthew@sys-con.com

Accounting

Financial Analyst:
Joan LaRose joan@sys-con.com
Accounts Payable:
Betty White betty@sys-con.com
Accounts Receivable:
Steve Michelin smichelin@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
Monique Floyd monique@sys-con.com
JDJ Store Manager:
Brunilda Staropoli bruni@sys-con.com

While at Sun Microsystems, **Kim Polese** was part of the Oak/Java team from 1993 on. As Java's original product manager, she led its 1995 launch. Kim left Sun in January 1996 together with Arthur van Hoff, Jonathan Payne, and Sami Shaio to cofound Marimba, Inc. As CEO, she led Marimba through a successful IPO and to profitability, and continued to serve on the board until BMC acquired it in 2004. She is now CEO of SpikeSource, Inc., the Kleiner-Perkins-funded opensource software company.

—continued on page 62

**MARCH
BUZZ**

Jtest®

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>



Ajit Sagar
Contributing Editor

SOA, MSOA, and Java

SOA is obviously the new buzzword of the day. Among the many acronyms, one that is seen very often is “Same Old Architecture.” In many ways, this is true. The key differentiator between the paradigms that have been prevalent in the past and this new incarnation of “service-orientation” is that the new definition of services is targeting the business as well as the technical side of the house. Same old architecture – different politics.

Mind you, I am not saying that this not needed. The processes and governance that has been formalized around SOA make for a very effective IT renovation roadmap. If you are interested in how SOA assists in IT renovation, check out the book *Enterprise SOA: Service-Oriented Architecture Best Practices* by Dirk Krafzig, Karl Banke, and Dirk Slama (The Coad Series). I recently published a review on this at my blog: <http://ajitsagar.javadevelopersjournal.com/read/1062164.htm>.

There is a tendency (propagated by vendors) for the industry to conclude that SOA means Web services. While Web services provide the ideal platform for implementing SOA, they are not the only option. Everyone automatically assumes Web services whenever the term SOA is mentioned. Well, adding Web services to SOA definitely gives SOA a bit of an oomph, and differentiates it from being the “Same Old Architecture.” After all, if you weren’t using Web services to implement SOA, what’s new about your solution?

While it is true that Web services offer a very attractive platform for realizing SOA, they are not the only technology available to do so. In fact, the main message behind SOA is not the “Web” but rather the “service.” The main objective of SOA

is to help organizations move toward a Service Oriented Enterprise (SOE). The main problem in organizations that SOA addresses is the ability to use architecture as a common tool for IT and business to achieve a common objective – IT agility.

I’ve found it a little surprising that while the term WSOA (Web Service–Oriented Architecture) is around the corner for everyone on the SOA bandwagon, no one really talks about MSOA (Message Service–Oriented Architecture). Although I’m sure I haven’t invented this term, I haven’t found it in my Google searches. You will find “Message Oriented Architecture” or “Messaging Service,” but not MSOA. Messaging is another way to leverage existing investments to realize SOA. It provides the protocol and the semantics. What it doesn’t provide right off the bat is a standard service registry akin to UDDI. So shouldn’t WSOA be a subset of MSOA? After all Web services provide a mechanism to exchange messages in a loosely coupled architecture and eliminate the tight coupling mandated by APIs.

As I was saying earlier, SOA is not the new concept – what’s new is its application. If you take away the whole message of “platform and language independence,” which never really happens when you actually implement something, is the concept any different from what Java proposed with Jini? In fact, I would say that Jini is one of the first software architectures that promoted the concept of SOA – although they made the mistake of adding the qualifier “network” to it. So it got associated with devices, not software applications. As the “computer” moved away from being the “network,” Sun’s message faded into burst bubbles of .com, and Jini slipped through the cracks.

When you look at Sun’s site, you wonder “Whither Jini?”

In the article “Jini Network Technology Fulfilling Its Promise” (http://java.sun.com/developer/technicalArticles/Interviews/waldo_qa.html) Jim Waldo has provided some very interesting insights into where Jini is today (“today” is relative since the article is about a year old). I found two statements in the article that talk about Jini from the SOA perspective:

- Probably the biggest misconception is that it is concerned primarily with devices.
- The message about devices hid the fact that Jini software is really a general service-oriented architecture.

It is all about the message. Sun has never really made any money on software. If Jini was marketed properly, it could have been the leading concept behind the new wave that is SOA. I remember back in 2000, in one of my previous lives, I had worked on a whitepaper where we borrowed the concepts from Jini and applied them to transactional marketplaces – the setup would be similar to what is touted in SOA today. Perhaps if Sun had used the word “messaging” instead of “network” and “service” instead of “device,” Jini would have evolved into the key architecture implementation for SOA today.

On a side note, it’s interesting to see how SOA has been globally accepted. I recently received an invitation to a Web services and SOA conference in China (details are available at www.ajitsagar.javadevelopersjournal.com/read/1088114.htm), where SOA seems to be gaining a lot of traction. This should be an interesting experience, and I hope to chronicle it in one of my future editorials. ☺

“Perhaps if Sun had used the word ‘messaging’ instead of ‘network’ and ‘service’ instead of ‘device’, Jini would have evolved into the key architecture implementation for SOA today”

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

Beyond Patterns: Thinking Objects

A social metaphor for writing software

by Bill Kohl

Algorithm: A detailed sequence of actions to perform to accomplish some task. –Webster

“Interaction is more powerful than algorithms.” –Wegner

Metaphor: Using a known idea to impart understanding of a new unknown idea.

Patterns and use cases have become accepted tools for creating OO apps. For many designers they are their only approach. But getting the full benefit of OOP requires a new way of thinking about creating object-oriented applications. It is called “Thinking Objects.” This article will offer a metaphor to help you understand and begin “Thinking Objects.”

Functionality

There are two ways of producing functionality in software that I would like to discuss: sequential (procedural, algorithmic) execution and interaction. People are familiar with sequential programming. It is the first programming paradigm taught and a natural way of thinking. It goes like this. For a given task (result), create a series of steps (and possibly sub tasks) and carry them out in sequential order. When the steps have been completed, the desired result is achieved.

Figure 1 shows the task of mowing the lawn. To perform this task we would carry out the sequential set of steps shown. After completing the last step we have mowed the lawn. Note that we have acted in a procedural way, carrying out a series of sequential steps. So, when accomplishing individual tasks, we think and act procedurally.

Interaction and the Division of Labor Metaphor

There is another paradigm for creating functionality; it is called interaction. Think about modern society. We live in an age of cars that think, man has gone to the moon, gene therapy, the Internet. It is a very complex society. And yet it runs smoothly while getting more complicated every day. If we were asked to simulate even a small part of this complexity with procedural programming, it would be a difficult task. How does our society do it?

We investigate this by looking at another example of achieving functionality: buying an automobile. If we wish to buy a car, we go to a dealership and meet a salesman who shows us cars. When we find what we want, we negotiate a price and the dealer has a contract printed for us to sign. Now we have to negotiate a loan, so we go to the bank and see a loan officer. Meanwhile, the auto salesman has applied for a title to our automobile. We return with loan in hand, giving the money to the salesman. He gives us the keys to the car and temporary title. We have achieved the functionality of buying a car. Notice that though we have carried out a sequence of steps, it has required the services of others to complete the process (see Figure 2). This is fundamentally different than mowing the lawn, which we did without any help. There we were able to act individually. Here, we act in cooperation with others whose functionality is carried out independently of ours.

Could we have bought the car without help? Would we have known how to create the sales contract, how to get temporary title, how to complete the paperwork and credit investigation to get a loan?

Probably not, but suppose we could have. What about the car, could we have designed it? Could we have built it? I think not.

Corporate Example

The example above lacks functional descriptiveness. So I want to offer another example of the second kind of functionality. In this example, the overall functionality produced is immediately identifiable; it is the goods and services provided by a corporation to its customers.

A corporation supplies a specific set of goods and services to its customers. These goods and services represent the corporation's functionality. Services represent obvious functionality while goods represent the functionality required to produce them. Let's look at how a corporation is structured to produce that functionality.

A corporation is an entity. It has facilities and employees. It has a mission and purpose. How does a corporation operate? It operates through its employees. Employees of a corporation have job assignments, and responsibilities that go with those job assignments. If all the employees carry out their responsibilities and these responsibilities have been assigned properly to achieve the corporation's objectives, it will be successful.



Bill Kohl works as a software architect for a large petroleum industry corporation. He has worked in software for more than 30 years, the last 15 in the OO world. He has been an OO instructor and mentor and a Smalltalk, C++ and Java developer and has extensive experience in developing object models for enterprise applications.

wkohl@houston.rr.com

Here is the part of the corporation that supplies the functionality of fulfilling a customer's order (see Figure 3). In this diagram, we see four corporate employees. They will cooperate to fill a customer order. Our corporation makes a product so a product will have to be assembled to complete the order. The ellipse represents the corporate boundary.

What happens when an order comes in? The order clerk gets the order from a customer. He or she has some responsibilities to fulfill, for example creating a purchase order, filing it, distributing copies, etc. After completing these responsibilities, he or she takes the order and asks the parts clerk to fill the order. The parts clerk pulls the parts, reconciles the inventory and passes the parts and purchase order to the assembly clerk, asking him to assemble the order. The assembly clerk assembles the parts, which then become the product. The responsibilities of the assembly clerk may include logging the product and giving it an ID number. He then passes the purchase order and product to the shipping clerk. The shipping clerk must determine and affix the postage and place the order for pick-up by UPS.

Here four employees have worked together to complete the customer order. Each has done their job in filling the order and has cooperated (interacted) with other employees to complete the process. This example makes clear how employees with responsibilities interact to supply an overall functionality for the corporation. In so doing, employees need only know how to carry out that part of the process for which they are responsible. (The idea of responsibility in object-oriented programming was put forward by Wirfs-Brock, et al.

If you haven't guessed by now, I have been describing what we know as division of labor, or in software terms, interaction or collaboration. Division of labor is as old as human society. It was not invented. It simply evolved as the best means of handling social complexity. Modern societies could not exist without it.

Flexibility

So division of labor evolved to deal with complexity in society and does an excellent job of it. What other benefits might it have? It happens to be quite adaptable. At one time in the history of medicine, a person who became a doctor was just that, a doctor, not a heart specialist or intestinal specialist, just a doctor, one variety. You went to a doctor and he did it all. Today we have medical specialists. But instead of having to redesign society's division of labor structure to accommodate a new medical hierarchy, the structure simply grew through specialization to accommodate the new complexity.

Division of Labor

Well, since this second kind of functionality has turned out to have some cool benefits like complexity management and flexibility maybe we should say a little more about it. We would like to know what makes it work so well in managing complexity, where does its flexibility come from and what is required for its operation?

How does it manage complexity? There are two kinds of complexity we are asking about here: data complexity and functional complexity. The data component is easily identified. It is all of the factual or data knowledge of our society. Such things as statistical knowledge, anatomical knowledge

and historical knowledge are data knowledge. The functional component can be thought of as all the processes that can be observed in a society, which we will call societal processes. Such things as mail delivery, automobile production, even sending men to the moon are examples of societal processes. From these, two types of process can be identified: discrete societal processes and complex societal processes.

Discrete societal processes are those that one person can do, whose performance is normally continuous, which have clearly defined stopping and starting boundaries and which can't be subdivided into smaller discrete processes. These would include such things as starting a lawn mower or typing a letter.

Complex societal processes are those that aren't discrete. Their completion depends on the execution of many discrete processes. Two distinct types of complex societal



Figure 1 Mowing the lawn

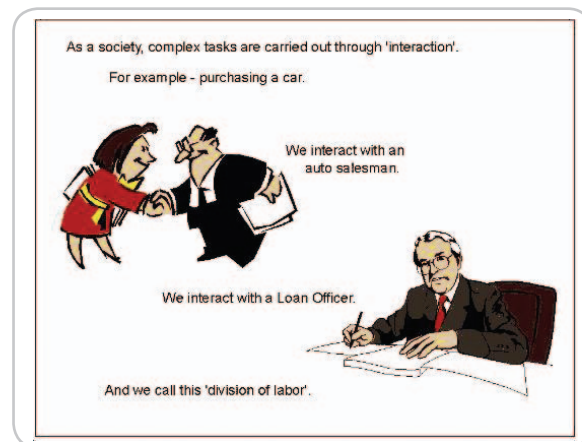


Figure 2 Division of labor

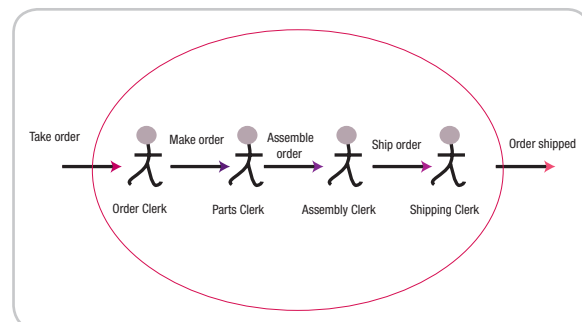


Figure 3 Fulfilling orders

processes can be identified: Enclosed and non-enclosed. An enclosed complex societal process is done by a single individual through a sequence of discrete processes. A non-enclosed complex societal process requires the collaboration of two or more people to complete. We must now ask, what is it that creates non-enclosed versus enclosed processes?

The answer is specialization. Society has “chunked up” discrete processes into what it calls specialties. Individuals in society become ‘Specialists’ within a well-defined, cohesive chunk of societal functionality called a specialty. Specialties include professions such as heart surgeon, auto mechanic or plumber. Functionally, specialties are composed of collections of related discrete processes. The purpose of a specialty is to identify discrete chunks of social functionality that single individuals can comprehend and master.

As we shall see, specialists form the basic unit for creating complex social functionality.

So far we have been discussing the functional component of societal knowledge. What about the data component? To see how the division of labor handles this, we first have to note that data and function are not inseparable (at least not when viewed through the division of labor prism). For example, think of the function required of a heart surgeon. He must have the process knowledge for performing heart by-pass surgery, heart massage, valve replacement and many other procedures. But these processes couldn’t be done without certain data knowledge including knowledge of human anatomy, human chemistry, human physiology and other specialized knowledge. In other words, the heart surgeon must have all of the data knowledge necessary to carry out the functionality required of the profession. The division of functional specialties defines the division of societal data. Each specialty found in a division of labor structure includes the data knowledge required to accomplish the functions required of that specialty. Now we can answer the question of how division of labor manages complexity.

A Complaint Resolution System

A phone utility wants to service its customers better by responding to every customer complaint. Its current system manually routes complaints from inception to completion and they sometimes get lost or shuffled to the bottom of the stack. As a result, response time is slow or non-existent. The company wants a new automated system that electronically routes customer complaints and tracks their progress, setting out an alarm when certain pre-programmed time limits on a complaint are exceeded.

Customers complain of such things as noisy lines, crosstalk on the line and leaning telephone poles. A complaint is first registered with the complaint department where it is given a priority. From there it is routed to maintenance. Maintenance schedules a repair date, and team and dispatches the team on the scheduled date with a repair order. When the repair is made, the team notes its completion date and time and any comments on the repair order. Finally, the customer is notified that the repair has been made and the complaint ticket is closed. You are to design and write the application.

In the business world there are two major kinds of apps you will run into: data maintenance apps and business process (workflow) programs. This app is the business process kind. The software is required to enforce and monitor the process of resolving a customer complaint.

Now ask yourself the question, “What specialists and coordination hierarchy would I need to accomplish this job?”

If I want to make sure no complaint is misplaced, lost or slowed down, I would want to assign every complaint to an employee whose only task is to push that complaint through the process and never lose track of it.

For a complaint to be treated as a discrete process in the utility company’s complaint system, it can be represented in Java objects as a `ComplaintProcess` Java class subclassed under a `Process` Java class. (For a discussion

of Process Objects see “Beyond Entity Objects,” JDJ, Sept. ’04.) A `ComplaintProcess` is a process object. But there’s data accompanying a complaint that we will represent as a `ComplaintForm` Java class.

It is important to note that the `ComplaintProcess` class is not a db class. In other words, there will not be a table representing the `ComplaintProcess`. (During a Thinking Objects session implementation details are not considered.)

An instance of the `ComplaintProcess` class, a `ComplaintProcess`, will be responsible for moving a customer complaint through the required business process. This corresponds to the horizontal level of coordination in a business model. Requirements for the `ComplaintProcess` class would include electronically routing itself through the complaint resolution process, maintaining forms related to processing the complaint, maintaining response time-outs with notifications if the complaint is slow in being processed, and closing out the complaint when the process is finished.

Each `ComplaintProcess` will have to deal with the customer service department and the maintenance department. Initially I would assume a façade object to interface to each of these departments. Since the façade object will only redirect client requests to other application objects, there would also be objects representing business logic required for interfacing with each department. So we may have a `CustomerServiceSupervisor` and a `MaintenanceSupervisor`. These objects would be responsible for creating forms associated with their department, creating database entries where required, etc. This suggests that there would be an object for each different form.

As a rule, developers put far too few classes in their apps. So don’t worry about adding new classes as you think of them. Besides, the `ComplaintProcess` object may want to delegate some of its responsibilities for timing to

a timer object. The timer, once started, would run until time-out when it notifies the `ComplaintProcess` that it’s associated with that it has hit time-out. Timers require a `TimerManager` responsible for creating new instances and re-instantiating existing instances from a database when necessary.

Here is the Java class list:

```

Process
ComplaintProcess
Timer
TimerManager
Form
ComplaintForm
MaintenanceForm
CustomerServiceSupervisor
MaintenanceSupervisor
MaintenanceScheduler
MaintenanceDispatcher
Vehicle
MaintenanceVehicle
Team
MaintenanceTeam

```

Of the classes above, only `MaintenanceVehicle`, `ComplaintForm`, `MaintenanceForm`, and `Timer` are likely to represent a database table. One should create a class for each database table. But these classes reside in the database translation layer and shouldn’t become part of the domain model. A domain model class whose only data was that of a database table, for example, the `MaintenanceVehicle`, would be represented in the database transition layer as `MaintenanceVehiclePersistentObject`. There are many other Java classes that would make up this program – for example, `Customer`, `Employee`, etc.

Enclosed complex processes are those that can be performed by a single specialist using only the discrete processes belonging to that specialty. Examples would include mowing the lawn or typing a letter. Since enclosed processes belong to a single specialist, that is a single individual, the functional pattern applied to produce these processes is the procedural or sequential pattern. This is a pattern we are familiar with so we have no trouble visualizing how this functionality is produced.

Non-enclosed complex processes require the collaboration of two or more specialists. Examples might include buying an automobile or building a house. It is with non-enclosed complex processes that we find a new pattern of creating functionality, one completely different from the procedural pattern. This pattern is called interaction or collaboration.

If we look at society in operation, we see specialists (individuals) making their knowledge (data and function) available to society as a whole by offering services to other members of that society. It is through the combination of services offered by sets of specialists that non-enclosed complex processes are realized. We can characterize this by saying that individuals in society (specialists) interact with each other by offering services to others and using the services offered by others to accomplish complex functionality. Division of labor implies, and to work requires, interaction of this kind among its specialists.

Interaction has two basic characteristics. The first is specialties, units of behavior that externally offer a specific set of cohesive services. The second is communication. Communication is required for one member of society (a single specialist) to request and get the specialized services of another member of society. It is required so that specializations can interact to create complex functionality.

(An observation can be made at this point about non-enclosed complex process knowledge. The truly complex behavior of society occurs through the interaction of groups of specialists. One may ask where the functional knowledge lies for these processes that span specialties. Let's take a complex processes like sending men to the moon. A process as complex as this requires more than simply interaction between sets of specialists, it requires coordination of this interaction. The responsibility for coordinating the task of sending men to the moon is delegated to a complex social entity, NASA. A complex social entity is made up of individual specialists. Some of these will be coordination specialists. Complex social entities provide services that require the interaction of many specialists. Thus, complex entities are formed to coordinate the interactions required to create complex functionality. Corporations are another example of these complex social entities.

To recap: Society uses division of labor to deal with complexity. Division of labor implies specialization. Specialization decomposes data and functional complexity into chunks that a single individual, termed a specialist, can deal with. These chunks are called specializations. A specialist's knowledge includes both factual (data) knowledge and process (functional) knowledge. Specialists offer their services to society in general. Complex social functionality is created through the interaction or collaboration of groups of specialists. This interaction implies communication so that members of society can request and get services from other members of society. It is this interaction between specialists of society that produces complex functionality such as landing men on the moon.

Paradigm Shift

We have seen that our traditional procedural method of creating functionality is not the only way to create functionality. There is another method we identified as division of labor in society, but is called interaction or collaboration in the software world. We have seen that division of labor manages complexity through specialization. We have also noted that division of labor is flexible and adaptable to ever-increasing complexity through subdivision of its specialization units. And finally we have noted that interaction between specialists produces all of the extremely complex functionality in society. We can relate all of this to object-oriented programming. Objects contain data and function corresponding to the knowledge and process of specialists. Objects present an interface through which other objects may use their services. Messages provide the means for objects to interact and thereby create complex functionality.

Object-oriented programming is said to require a paradigm shift. That shift is from procedural programming to interaction programming. Object-oriented programming applies interaction to create modern complex software. Objects (specialists) and messages (communication) are used to create software that is more maintainable and extendible due to its inherent flexibility.

Conclusion

OOP requires a paradigm shift from procedural thinking to interaction thinking. This transition is not an easy one. Procedural thinking is natural to humans and the habit is not easily modified. We have not mentioned what language tools would be necessary to implement the division of labor pattern. It would require encapsulation of data and function to create specialists. It would require a communication medium between specialists such as messaging. Identity would be necessary for directing messages to the proper specialist. Sub-classing and inheritance (sub-specialization) add to the flexibility as does polymorphism.

A Note on Object Specialists

Object specialists are different from societies specialists. Object specialties are those needed to create a software application. We may say that objects are specialists due to the cohesive functionality they encapsulate. Object-oriented apps have been referred to as societies of collaborating objects. In this sense they exhibit many of the same characteristics as human specialists, interacting with one another to produce the complex behavior needed in an app. Yet, it is still possible to produce spaghetti OO code just as it is to produce spaghetti procedural code. Coupling can be minimized with coordination specialists, such as NASA, whose responsibility is to decouple objects from objects and encapsulate processes in themselves (see "Beyond Entity Objects: Modeling Concepts with Objects," Java Developers Journal, Sept. 2004) and by the judicious use of patterns. ☺

Resources

- Wegner, Peter. "Why Interaction Is More Powerful Than Algorithms," Communications of the ACM, May 1997 (<http://www.cs.brown.edu/people/pw/>)
- Wegner, Peter, "Interactive Foundations of Computing," Final Draft, Theoretical Computer Science, February 1998 (<http://www.cs.brown.edu/people/pw/>)
- Wirfs-Brock, R. *Object Design*, Addison-Wesley, 2003. (<http://objects.bydesign.com/>)
- Mullin, Mark. *Object-Oriented Program Design: With Examples in C++*, Addison Wesley Longman, Inc, August 1989

Development of Component-Oriented Web Interfaces

by Alex Maclinovsky & Alexey Yakubovich

A case for the Vitrage Framework

The latest trend in information portals and Web applications has been to build complex Web pages. To present large amounts of information and functionality without compromising usability, designers have imposed a clear structure by grouping related elements together. Such cohesive, visually distinct constructs, or *compartments*, often with their own presentation logic, have become an essential feature of complex Web applications.

The first half of this article introduces the notion of compartments as a fundamental concept that pervades the layers of many Web applications and stages in their lifecycle. It defines compartments and analyzes their structure and key characteristics. Then it will examine the existing presentation layer technologies for building compartmentalized applications and demonstrate their weaknesses.

The second half of the article proposes an architecture and framework that directly support design and development of compartmentalized applications and improve their performance. It demonstrates how the use of this framework improves developer productivity, facilitates reuse and yields more flexible and maintainable applications.

the Web; a quick survey of a number of prominent Web sites shows that many of them use easily identifiable compartments as primary structural elements.

Definition

Despite visual and structural diversity, all compartments share fundamental elements. We define a compartment as a rectangular area on the page that presents dynamic content and has the following characteristics:

- It has regular structure and layout
- It contains related content elements
- It's visually and functionally distinct from the rest of the page
- It has internal presentation logic (i.e., layout depends on the content displayed)
- It behaves atomically within the page.

To illustrate this definition, these are some of the compartments that can be seen on the screenshots in Figure 1:

- On Travelocity – Great Getaways, Fare Watcher, Cruises & Vacations and Travel Tools.
- On FirstGov – Agencies, Information by Topic and In Focus.

The following elements don't meet the definition and wouldn't be considered compartments:

- The grey left navigation bar on the BBC site because it's static
- "Find me the best priced trip" on Travelocity because it's static and doesn't have regular structure or presentation logic
- "E-mail This Page" on FirstGov and the round navigation buttons at the top of the Yahoo page (i.e., Personalize and Finance) because they lack regular structure or presentation logic.

Structure and Behavior

The Programs & Campaigns (P&C) Compartment presented in 0 came from one of the systems developed by the authors. It will be used as an example in this article.

Typically a compartment consists of a header, footer and a body. The header and footer together define the visual elements, including frame, background, title and global links such as "[All Programs](#)" and "[All Campaigns](#)" and the other elements that belong to the entire block.

The body consists of a series of *contentlets*, elementary units of dynamic content presentable on a Web page, separated by optional spacers or other structural elements. A contentlet has a number of core attributes: Name, Description, Image and Display Order. Additional attributes, such as Date, Type and Source, may be represented as needed for a particular problem domain or implementation. All these attributes are optional.

One important type of contentlet is a *Content Reference*, which adds another core attribute, a URL. Any dynamic content targeted to a compartment can be represented as a collection of contentlets.

In contrast to static page elements, a compartment has intrinsic behavior, which is governed by its presentation logic. Presentation logic is a set of rules determining how to render each contentlet, which contentlets are considered renderable and how to render the entire compartment, depending on the collection of contentlets it receives. Such rules may include:

- Lay out contentlet attributes in four columns in order – Image, Name and Description, Type, Date
- Consider a contentlet renderable if one of the core attributes is defined



Alex Maclinovsky is

a solutions architect with Roundarch, Inc. For the last 15 years he has focused on developing and architecting large distributed object systems on enterprise, national and global scales. His professional interests include solution-oriented architectures, adaptive frameworks and OO methodologies.

amaclinovsky@roundarch.com



Alexey Yakubovich works as a

framework architect at Roundarch, Inc. He received his Ph.D. in mathematics in Moscow State University for research in mathematical logic, and has published more than two-dozen articles in mathematical magazines. Alexey has spent 20 years in software development.

ayakubovich@roundarch.com

Introducing Compartments

Figure 1 includes screenshots from some well-known Web sites. These sites all have a different look-and-feel from one another, serve different purposes and address different audiences. However, they use the same approach to organizing and presenting a wealth of information in regular, visually distinct blocks or compartments. Such compartmentalization is common throughout

Configurations for Embedded Reporting

Scaling Your Integration to Different Server Environments

Flexibility in configuration is essential for any embedded reporting solution. Developers must have the foresight to consider a reporting solution's value to both current and future projects. Because different environments and applications call for different configuration methods for report integration, developers must seek a solution that provides the flexibility to integrate in any environment.

With over seven years experience meeting the reporting demands of organizations worldwide, JReport has been deployed in a range of environments from single server to server clusters.

Following are three examples of common configurations for embedding reporting into an application, and reasons why an organization might select any particular method.

Single Server Integration

In environments where the application architecture requires a level of protection or isolation from various application components, a reporting solution could be deployed as a stand-alone solution on the same server as the application. The reporting solution would be maintained in a separate JVM to help ensure reliability.

In this environment, the application's JSP or HTML page invokes the reporting server via HTTP or RMI calls. Results are returned to the application as HTML or DHTML reports.

Some JReport users prefer single server integration because it provides good performance and is easy to manage. However, server resources like CPU and memory must be allocated, potentially resulting in unbalanced resource utilization.

Tight Application Integration

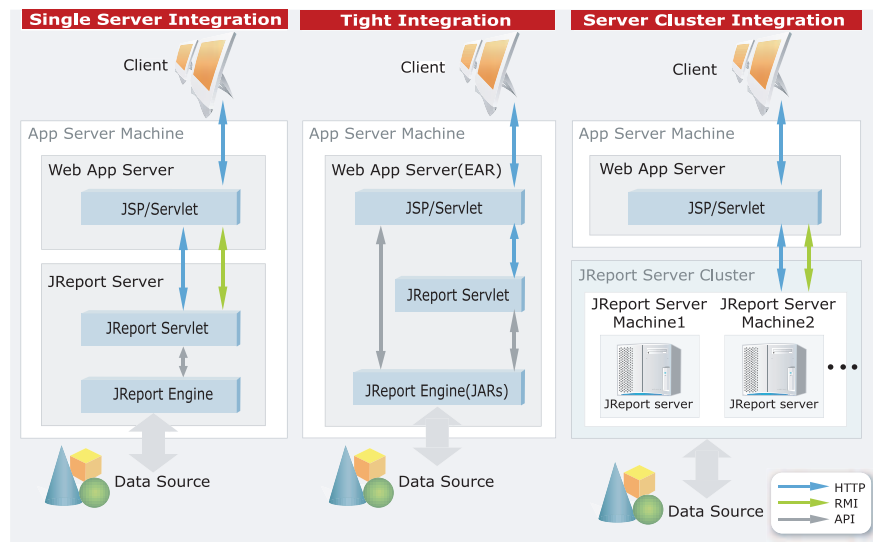
Tight integration addresses the allocated server resource issue by embedding the reporting tool directly into the application. This is best achieved with a 100% J2EE-based reporting tool which can be packaged into one or more JAR files that are then included in the application's EAR file. A set of APIs are used to run reports, and tag libraries can be utilized to embed reports into an application's JSP

In this approach, the reporting tool and application will share a single pool of server resources. In addition, only one JVM and server must be managed.

Most independent software vendors (ISVs) prefer a tight level of embedding because there is a physical integration which allows the application and JReport to be deployed as one product. And, JReport's small footprint helps keep total file size manageable.

Selecting a Configuration

Functionality is not compromised by any of these embedding configurations. The main differences are the physical location of the reporting server, how that server is invoked, and whether the integration is logical or physical. Because each configuration method has its advantages, you should carefully analyze your project and deployment requirements.



Server Cluster Integration

In some environments, it is desirable to have dedicated servers available to run reports. For example, a leading financial institution uses JReport to run over 100,000 reports daily, each of which can contain tens of thousands of rows of data. To ensure successful, fast report generation and delivery, even during periods of peak report activity, JReport was deployed in a server cluster environment.

By scaling server resources, extremely heavy workloads are completed in as little time as possible. Communication between the application and JReport in this configuration is achieved via HTTP or RMI calls.

This method requires expertise in network configuration and a larger investment to acquire and maintain the hardware involved. But, if running high volume, resource-intensive reports is your priority, the best option is to select a reporting solution with a proven ability to scale to large server clusters.

A reporting solution like JReport provides you with the flexibility you need to meet all your embedded reporting needs now and in the future.

Get a Free Reporting Requirements Evaluation from a JReport Expert

With thousands of deployments worldwide, Jinfonet Software understands the advantages and disadvantages of different deployment configurations. We have helped leading organizations in virtually every industry choose the embedding architecture that best fits their environment. Let us help you identify the configuration that makes the most sense for your project.

For a FREE reporting requirements evaluation or a trial of JReport, please contact one of our reporting experts today.



JReport is a 100% Java reporting solution that leverages J2EE industry standards and modular components to deliver a solution that easily embeds into any application. With JReport, report development and deployment in diverse environments no longer require costly integration of various resources from different vendors, various data sources, and legacy systems.

Call 301-838-5560 or visit www.jinfonet.com/jp3.htm

Jinfonet Software
(301) 838-5560
www.jinfonet.com



Figure 1 Compartments on familiar sites

Page	URL	Compartments
Yahoo home page	http://www.yahoo.com/	19
Excite home page	http://www.excite.com	16
Carsdirect home page	http://www.carsdirect.com/	11
Amazon entry page	http://www.amazon.com	23
FirstGov for Business	FirstGov.gov	11
CDC home page	http://www.cdc.gov/	5
Illinois State home page	http://www.il.gov/	11
Travelocity home page	http://www.travelocity.com/	8
Delta home page	http://www.delta.com/	9
Deloitte careers page	http://careers.deloitte.com/	7
BBC News	http://news.bbc.co.uk/	17

Figure 2 Summary results of compartmentalization survey

- Remove the Date column if none of the rendered contentlet defines it
- Change the compartment title to “Programs” if all contentlets are the *Type program*;
- Remove the entire compartment if it doesn't have any renderable contentlets.

It's often difficult to demarcate the exact boundary between business logic and presentation logic; this is a subject of hot discussion in the development community¹. A key distinction is that business rules exist in the problem domain, while presentation logic is pertinent only to a computer applica-

tion. Nevertheless, presentation logic usually comes from business requirements and is equally important in building successful software. However, in contrast to the amount of attention paid to application business logic, there is considerably less support for implementing presentation logic in Web-based applications.

Role in Application Lifecycle

The role of compartments isn't limited to system design and construction. Normally, compartments are identified early in the development process and used to construct wireframes, capture functional requirements

and design prototypes. They also play an important role in testing as a basis for individual test cases.

Presentation Technologies

There are a number of JSP-based technologies currently available to develop the presentation layer of compartmentalized Web applications. Portlets, different flavors of templates and tag libraries are commonly used. Each has inherent strengths and weaknesses.

Portlets are the most direct realization of compartments, but they can only be used in a portal platform. Using a portal platform is not always appropriate because of the cost, performance or technical limitations in the URL structure, as well as problems with *bookmarkability*, navigation, the lack of visibility by Web search engines, layout restrictions and other issues.

Template approaches like SSI, *ColdFusion* and *Tiles*² make page layouts flexible. They help to reduce the complexity of JSPs by breaking them into smaller, more manageable fragments that can be reused between pages. This decomposition can be taken to the level of individual compartments.

However, such a mechanism can only be applied to identical JSP fragments; it can't be reused for similar compartments. Although templates can simplify the maintenance of small JSP applications, advantages disappear as the number of pages and compartments on each page grow and developers have to manage thousands of individual JSPs.

Neither of these approaches addresses the rendering of dynamic content or the implementation of presentation logic inside the compartments, leaving the developer to rely on tag libraries.

Custom Tags as the Prevalent Choice

Custom tag technology has become the most prevalent tool in developing dynamic JSP user interfaces. It gained acceptance because it promised to separate the presentation from the content, eliminate Java programming from JSPs and let HTML designers use familiar tag syntax.

Tag libraries work really well in many cases, but become less usable in more complicated cases, particularly when JSPs need to implement the non-trivial presentation logic often required by compartments. That's why publications promoting tag libraries never go beyond the simplest examples³. Sun's implementation of Pet Store illustrates this point perfectly. Pet Store serves as a blueprint for building applications using J2EE technolo-



21 WAYS TO USE SPREADSHEETS IN YOUR JAVA APPLICATIONS

A free offer for readers of *Java Developer's Journal*!

Formula One e.Spreadsheets Engine:

Finally, there's a *supported*, Pure Java tool that merges the power of Excel spreadsheets and Java applications.

- 1 Automatically generate dynamic Excel reports. No more manual querying and cutting-and-pasting to create Excel reports!
- 2 Manage calculations and business rules on J2EE servers with Excel files. No more translating Excel formulas to Java code!
- 3 Embed live, Excel-compatible data grids in applets and Java desktop applications. No more static HTML or presentation-only data grids!

Download this quick-read white paper and trial today!



Download your trial and test our demos and sample code. See for yourself how the Formula One e.Spreadsheets Engine can help your Java application leverage the skills of Excel users in your business.

<http://www.reportingengines.com/download/21ways.jsp>



888-884-8665 • www.reportingengines.com
sales@reportingengines.com

Need to deliver reports from your J2EE application or portal server? Try the Formula One e.Report Engine!



Build reports against JDBC, XML, Java objects, BEA Portal Server logs, BEA Liquid Data, and other sources visually or with Java code. It's embedded! No external report server to set up. Unlimited users and CPUs per license.

<http://www.reportingengines.com/download/f1ere.jsp>

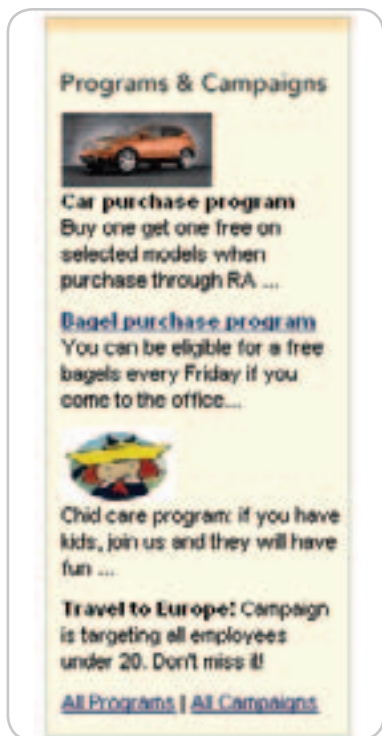


Figure 3 Programs & Campaigns Compartment

gies like JSTL tags, but the most complex table is limited to static columns and doesn't have any presentation logic.

A Real-Life Example

To see how this works in the real world, consider the P&C Compartment presented in 0 that came from an actual application.

The compartment is designed to present a collection of contentlets arranged in a one-column table, one object per cell. Contentlets are considered renderable as long as they have one core attribute defined. The following requirements should also be satisfied:

- Invalid and unrenderable contentlets should be omitted
- If no contentlets were rendered, the entire table with its header, footer, frame and background should not be rendered
- Visual attributes should be arranged vertically in the following order: image, followed by name, then description – if an attribute is undefined, it should be omitted.

If the contentlet is a Content Reference, these additional rules apply:

- If an image is present, it should be rendered as the link
- If there is no image, the name should be rendered as the link
- The description is never rendered as a link.

These requirements are quite modest, and the compartment occupies only a small part of a page. The table below contains key metrics summarizing implementations with BEA portal tag libraries and with Sun's JSTL, illustrating the complexity of implementing even the simple P&C Compartment using custom tags. The development was done by experienced Web programmers.

The actual JSP code is too complex to be presented here although it's available online, along with a more in-depth analysis in the form of a whitepaper. This implementation would not be significantly simplified by using portlets or Tiles templates.

Using this approach to develop a page with 10 or more compartments will result in thousands of lines of JSP code, making it highly inefficient for implementing applications similar to the ones presented in Figure 1.

Why Tags Fail

The reason custom tags lead to such complexity is that tag libraries are designed to cover the basic HTML constructs like anchors, tables and forms.

They are effective when the layout doesn't depend on content. The later tag library extensions supporting loops, conditions and other control flow constructs inside a JSP (i.e., iteration, choice and choiceMethod) have very limited capabilities and poor expressive power for programming presentation logic.

In other words, tag libraries force developers to program in a primitive language that lacks clear structure and doesn't allow code reuse beyond simple includes and cut-and-paste. To complicate things, there are many incompatible flavors of this language, each with its own quirks and limitations. For example, BEA's netui-data library let one use choice statements only inside a loop, while a c:url tag in a JSTL doesn't support named anchors.

It has been firmly established that a JSP is not a good place for exercising

imperative programming, be it java servlets programming or tag programming. A new approach is needed that can solve the problem of developing dynamic compartmentalized applications effectively.

Success Criteria

How would you measure the success of this new approach? Previous analysis has helped formulate key criteria for its success.

1. **Direct support for key abstractions** from the problem domain – compartments, contentlets and Presentation Logic.
2. **Produce clean, simple, easily maintainable JSPs** free of any logical programming.
3. **Use appropriate means to express different presentational aspects** – let programmers implement logic and data manipulation in Java, while allowing Web designers to define visual design using familiar syntax and tools.
4. **Facilitate both physical and logical reuse**, allowing presentation components to be used in multiple places and combined into more complex ones, as well as reuse of common functionality through inheritance and delegation.
5. **Increase developer productivity.**
6. **Ensure compatibility** with a wide variety of existing J2EE platforms, MVC architecture and mainstream frameworks.
7. **Remain complementary to existing presentation technologies**, and allow combining them when appropriate.

Vitrage Framework

In developing large information portals, we have faced all the challenges described in the first half of this article. Having experienced the disappointments of existing approaches, we set out to develop a new solution that would meet the criteria outlined above. The result is called the *Vitrage* Framework, vitrage being the French word for stained glass. Vitrage is a solution centered around the notion of *JSP-blocks*. A JSP-block is the Java realization of compartment abstraction.

Architecture

The diagram in Figure 4 presents the structural components of the Vitrage Framework. It contains three major com-

Vendor	Number of lines	Number of different tags	Individual tag occurrence	Level of tag nesting
BEA	129	9		29 3
Portal				
JSTL	105	7		24 3

Overview of Sun Java Studio Enterprise 7

by Ashwin Rao, Robin Smith, and Marina Sum

In December 2004, Sun released Sun Java Studio Enterprise 7 (called Java Studio Enterprise 7 for short), a robust enterprise development platform that spans the entire life cycle of building, debugging, testing, deploying, and tuning Java 2 Platform, Enterprise Edition (J2EE) applications, including portal components and Web services. Java Studio Enterprise 7 boasts much more than a development tool set. Its impressive array of features and services simplify the increasingly complex tasks of software programming. The entire development process—from architecture to design to code development to deployment—is rendered seamless and intuitive.

This article describes the Java Studio Enterprise 7 capabilities and outlines its pricing model.

Capabilities and Service Offerings

The Java Studio Enterprise 7 interface is based on the NetBeans 3.6 IDE, renowned for its user-friendly, full-featured environment in the open-source community. This section highlights the major capabilities in Java Studio Enterprise 7.

Instant Developer Collaboration

With Java Studio Enterprise 7, development teams can communicate and dynamically collaborate online, regardless of location. That is, they can efficiently and seamlessly share different types of information from two collaboration channels:

- Chat channel — Team members can send text messages to one another in one of five formats: chat, plain text, Java code, HTML, or XML. Java Studio Enterprise 7 automatically detects the message format and, in the case of code, displays the message, complete with syntax highlights and line numbers. In addition, the chat window is fully code-aware; code completion and unobtrusive Javadoc pop-up windows provide easy reference points for communicating ideas.
- File sharing channel — Team members can share and edit files. Participants can work on a file simultaneously; automatic locks on edited lines prevent the same part of a file from being updated at the same time. Consequently, from this channel, a team can cooperatively write code, fix bugs, and explore new ideas as if one keyboard serves all. Moreover, Java Studio Enterprise 7 keeps the shared files distinct from their originals, thus maintaining the safety of the code.

Integrated Modeling with Unified Modeling Language

Integration of the Unified Modeling Language (UML) in Java Studio Enterprise 7 complies with that language's latest standards and its upcoming 2.0 specification, lending efficiency and ease in software architecture and code generation. Altogether, you can work in three modes: analysis, design, and implementation. The interface for the diagrams, objects, attributes, and relationships makes for smart navigation and code manipulation. Here are the benefits at a glance:

- Java Studio Enterprise 7 generates source code from and incorporates code edits into the model. That way, the source matches the model in real time without your having to manually mark or tag the source.
- You can reverse-engineer code, creating interaction diagrams that reflect the overall structure, for analysis, revisions, tests, or brainstorming.
- Available in the Design Center is a catalog of patterns—helpful templates, including Enterprise JavaBeans (EJB) components, for building models. You can create and catalog patterns of your own.

Choice is the ultimate, however. Feel free to turn off code generation and use Java Studio Enterprise 7 for design only.

- You can easily document projects in Java Studio Enterprise 7. From the code, Java Studio Enterprise 7 can export text and graphics to HTML and thus transmit implementation details, properties, and relationships to your work group.

Performance Tuning

Performance tuning in Java Studio Enterprise 7 enables you to tune applications early in the development cycle. You can then detect and correct architectural deficiencies earlier, resulting in less rework in the subsequent phases of development and a reduction in cost.

With Java Studio Enterprise 7, you can fine-tune your applications with an integrated profiler, called Enterprise Application Profiler, accompanied by a load generator. Enterprise Application Profiler captures the application's transaction details and presents them in tabular or graphical form. You can save, revise, or redisplay the data—valuable features for subsequent load generation and testing.

Furthermore, you can analyze an application's design and behavior by monitoring its method-level performance, pinpointing problematic spots, and saving the data as a results tree for future visits. It's your choice as to what and how much to monitor. You can selectively instrument the methods in any or all of your deployed applications.

Refactoring

With refactoring in Java Studio Enterprise 7, you can easily maintain large code bases. For example, you can rename and thus better identify code components. In addition, you can revise the occurrences of the renamed components, inspect the code base with the find capabilities, or optimize your code by extracting program constructs.

Here are the most common refactoring tasks you can perform by following short procedures in the Java Studio Enterprise 7 interface:

- Look up how to use classes, fields, methods, constructors, and variables.
- Rename constructs.
- "Bean-ize" fields.
- Move classes, fields, or methods to another class or package.
- Extract code segments to separate methods.

Usability

Java Studio Enterprise 7 is easy to install as a single unit. The integrated tools and services require no separate installations or configurations.

The look and feel of Java Studio Enterprise 7 epitomizes speed, convenience, and intuitiveness. The property windows boast a clear display of object attributes, and wizards guide you through tasks with clear feedback messages. Moreover, you can drag and drop files and other objects between task-oriented windows.

Developer Services

Seldom is software development a one-person project. By and large, structured team efforts can benefit from interactions, networking, and discussions with the developer community far and wide.

As a Java Studio Enterprise 7 subscriber, you are also a Sun Developer Network member. That membership entitles you to email support and participation in online product and technology forums at which to liaise with peer engineers at Sun and outside of Sun. Not to be missed are also Web chats, code camps, audiocasts, Webinars, and in-depth technical articles on valuable and helpful topics—implementation and integration stories, tips, newsletters, white papers.

Another offering is Enterprise Software Express, which delivers the latest work-in-progress code base from the Java Studio Enterprise 7 development team before a formal release. By reviewing the product and commenting back to the team, you help shape its design and behavior.

Integrated Server Software and Other Tools

Java Studio Enterprise 7 comprises this integrated set of server software from Sun:

- Sun Java System Application Server
- Sun Java System Web Server
- Sun Java System Portal Server
- Sun Java System Directory Server Enterprise Edition
- Sun Java System Access Manager
- Sun Java System Message Queue

Taking advantage of this software ensures simplicity of access, operations, and management of your applications.

Java Studio Enterprise 7 also includes these two handy tools:

- Sun Java Studio Web Application Framework, a graphical environment that comprises reusable components for building highly scalable J2EE applications. Of note is the Model-View-Controller (MVC) design pattern that distinguishes an application's presentation logic from its business logic.
- Portlet Builder, a plug-in with which you can create portal components, called portlets, based on the JavaServer Pages (JSP) technology in compliance with Java Specification Request (JSR) 168. Be sure to consult the collection of prebuilt portlets in the package and their source code: calendar, address book, mail, search, and others.

Pricing

Java Studio Enterprise 7 is available at a per-developer, per-license price of \$1,895 per year. This price also includes a year's subscription to Sun's developer services, during which you're entitled to online technical support and free upgrades that may become available during that period. You can renew the license for \$1,325 per year.

Java Studio Enterprise 7 is also available as part of the new Java System Suites at www.sun.com/software/javaenterprisesystem/suites/index.xml#suites.

Conclusion

Java Studio Enterprise 7 makes the software development process a smooth sail, resulting in productivity, effectiveness, and a transparent and cooperative work flow for engineering teams. We look forward to reading your feedback and to welcoming you on board.

For a free, 90-day evaluation download, go to developers.sun.com/prodtech/javatools/jenterprise/downloads/.

About the Authors

Ashwin Rao, a senior product manager in the Developer Platforms group at Sun, focuses on programming tools for J2EE architecture, development, and Web services. A software development veteran of almost nine years, Ashwin began his career by building real-time, embedded, and distributed systems in the defense industry. Before joining Sun, he led an engineering team on next-generation application platforms and tools at Baan Labs, an R&D group at Baan Company.

Robin Smith, a product line manager for Sun Java Studio Enterprise 7, managed the development and delivery of Sun Java Studio Web Application Framework, Sun ONE Integration Server, and Sun Java Studio Application Server. Before coming to Sun, he worked as a software engineer and architect for two decades and designed and developed numerous scalable and robust commercial applications. He then became development director at Precise Connectivity Solutions for Platform Adapter Component (PAC) middleware. Robin is a recognized authority on the IBM VM/370 and AS/400 operating systems and has developed several major, related system management products that were distributed and sold worldwide.

Marina Sum is a staff writer for Sun Developer Network. She has been writing for Sun for 15 years, mostly in the technical arena.

ponents: *HTML Code Generation*, *Vitrage Container* and *Development Tools*. The HTML generation itself is implemented on three levels: formatters, renders and blocks. Each layer uses components of lower layers and adds some new functionality.

Formatters

Formatters are used to generate elementary HTML tags with parameters at runtime. Tag parameters can specify layout attributes (such as background, border and span), as well as functional attributes (such as name, value and href). Formatters provide the low-level HTML-specific structure for JSP-blocks.

Renders

Renders use formatters as building blocks. There are three layers of renders: *content renders*, *HTML element renders* and *Composite renders*. All renders implement an interface *ARender*. This interface has two key methods:

```
String build(Object o, ICondition c);
void setLayout(ILayout l);
```

The method build() is invoked to

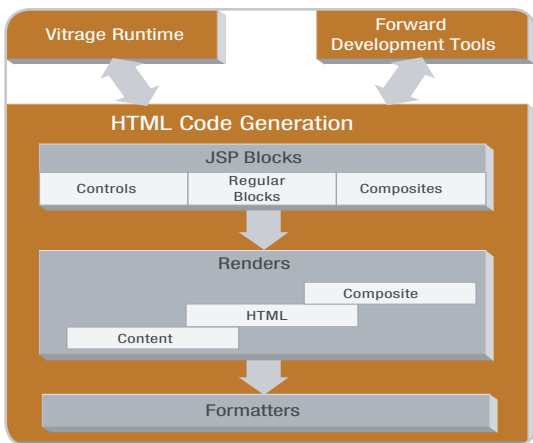


Figure 4 Architecture of the Vitrage Framework

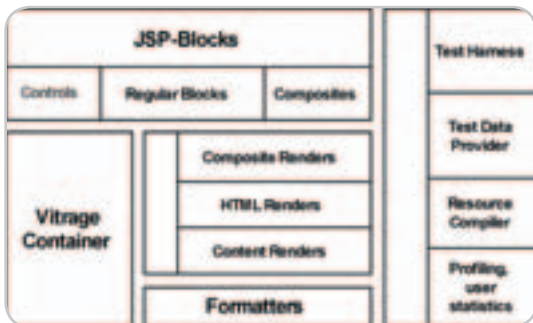


Figure 5 Architecture of the Vitrage Framework

generate HTML fragment, the method setLayout() is used to specify layout parameters for a render.

Content renders are responsible for presenting actual content. Currently, this category includes several flavors of contentlet renders. Vitrage implements a number of content renders with internal managers or *Oracles* that cover a variety of presentation strategies for contentlets. Another useful type of content render is Document Render, which can be used to embed an entire file into a page. Additional domain-specific renders can be implemented if required.

HTML renders use content renders to arrange content into a required HTML structure. Vitrage contains a broad selection of HTML renders including CellRender, RowRender, TableRender and SpanRender. FlatRender plays a special role; it's used to insert a static HTML fragment, usually obtained from the resource bundle, into the generated HTML structure. For instance, when an HTML table is generated, the FlatRender can provide a spacer between rows.

Composite renders like SerialRender and ConditionalRender logically combine other renders.

The SerialRender serves as an ordered container of other renders that invokes them sequentially. The ConditionalRender contains a collection of pairs of objects. Each pair consists of a render object and a condition object that implements the interface *ICondition*. The ConditionalRender in the build() method iterates over each pair, checking if the condition is true, and for the first of such pairs, invokes a build() method on a corresponding render. Only one render is invoked. If no condition satisfied, the ConditionalRender returns an empty string.

The following code fragment demonstrates assembling standard Vitrage renders into a custom render for the JSP-block that implements the P&C compartment described above.

```
1 public ARender prepareRender(
2     IBlockData data)
3 {
4     CntRefRender crr =
5         new CntRefRender(BASE_ALIGN);
6     crr.setImageCtrParam(true,
7         true, true);
8     crr.rightHanderBlock();
9     crr.setLayout(new ConRefLayout(
```

```
10     cssClass, 0, 0, 0));
11     CellRender cr =
12         new CellRender(crr, BASE_ALIGN);
13     RowRender rr = new RowRender(cr,
14         BASE_ALIGN);
15     ConditionalRender spr =
16         new ConditionalRender();
17     spr.addRender(new FlatRender(rs),
18         new NotRslv(new FirstElemRslv()));
19     SerialRender mr =
20         new SerialRender();
21     mr.addRender(spr);
22     mr.addRender(rr);
23     return mr;
24 }
```

At the top of the hierarchy is the main render object **mr** of class *SerialRender*. That render contains two renders: **rr**, which renders the actual rows, and **spr**. The second one is responsible for spacing between rows of the result table. It is a conditional render that will include a spacer HTML fragment defined in **rs** for all rows except the first one. The value is read from a resource bundle at startup. The first row in the result table doesn't require a spacer. The object of class *FirstElemRslv* recognizes the first row in a table and the result is negated by object *NotRslv*.

The second part of the main render is the chain of *RowRender*, *CellRender* and *CntRefRender* renders. These renders draw row tag, cell tag and the content reference object representation correspondingly. The following diagram illustrates how a block render is assembled from standard renders.

Let's consider how a render processes each contentlet.

When method build() is invoked on a render, it normally creates a *StringBuffer* and generates some HTML code into that buffer. For instance, if the main render is a *RowRender*, it adds an opening tag `<TR>` to the buffer with attributes specified in its layout object. Then the render invokes build() method on all contained renders in turn, passing along the contentlet. It adds the results from the containing renders to the buffer; then performs necessary post-processing (i.e., adds a closing `</TR>` tag and returns the content of the buffer). The outer render can always intercept and modify results of its inner renders.

JSP-blocks

The top level of code generation

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.



is provided by JSP-blocks. All blocks implement the IBlock interface with the method:

```
String build(HttpServletRequest r);
```

That method is invoked at the time of rendering from the JSP.

The Vitrage Framework implements three types of blocks: *Controls*, *Regular blocks*, and *Composites*.

Controls are the simplest kind of blocks to extend class ABlock directly and have to provide their own implementation of method build(). Typically they don't render contentlets but are used to create smart titles, pagination controls and A-to-Z indices, for example.

Regular blocks are responsible for rendering collections of contentlets in compartments. They encompass probably more than 90% of all blocks in a typical application. Regular blocks extend abstract class SimpleBlock. They should implement only two methods:

```
IBlockData getData(HttpServletRequest r, String gn);
ARender prepareRender(IBlockData d);
```

Method getData() is responsible for obtaining data, normally from the request object. Typically, IBlockData would contain a collection of contentlets

and other supplementary data, such as the title of the compartment. Method prepareRender() assembles a block-specific render from standard renders.

Class SimpleBlock implements method build(). This implementation generates the block's header, then iterates over contentlets passed in the IBlockData object and generates the footer. While iterating over contentlets, it passes each one to the block's render, accumulating the result and recovering from any exceptions. If one contentlet was rendered successfully, build() returns the entire HTML; otherwise, it flushes the buffer and returns an empty string.

Such architecture allows implementing presentation logic on multiple levels: for each contentlet inside the block's render and for the entire block in build() method. Composite blocks implement such logic on an inter-block level.

This design facilitates a high degree of code reuse among blocks. Figure 7 contains a complete implementation for a class that renders the P&C compartment with all the business requirements listed above:

```
1 public class PAMProgramsCompaigns
2   extends RegularRightBlock
3 {
4   public PAMProgramsCompaigns(
5     String title, String blockID)
6   {
7     super(title, blockID);
8     setEndFrag(
9       getResource(blockID, END_FRAG));
10    setCssClass(
11      getResource(blockID, CSS_CLASS));
12    rowSpacer = getResource(blockID,
13      ROW_SPACER);
14  }
15 }
```

Class PAMProgramsCompaigns inherits from RightHandBlock, as shown in **Error! Reference source not found. Error! Reference source not found.** contains completes the RightHandBlock class implementation.

```
1 abstract public class RightHandBlock
2   extends CTSearchEntryBlock
3 {
4   String rowSpacer;
5   public RightHandBlock(String title,
6     String id)
7   { super(title, blockID); }
8 }
```

```
9   protected String getStartFrag()
10  {return getResource(id, FSTART);}
11
12  public ARender prepareRender(
13    IBlockData data)
14  { // see Figure 5 }
15 }
```

All classes above PAMProgramsCompaigns are reused by other blocks in the application and don't contain any code specific to the P&C compartment. As a result, the whole compartment implementation takes just 10 lines of code.

This example clearly demonstrates that code reuse in the Vitrage Framework can make a block implementation very simple. These examples were taken from an application that actually contains a dozen compartments that differ only slightly from the P&C compartment. With Java inheritance, implementing a slightly different compartment can be accomplished with a small number of Java statements. As discussed above, with custom tags, it would probably take another 29 tag occurrences and 129 lines of JSP code for each compartment.

The JSP fragment that invokes the block looks as simple as:

```
<blocks:useBlock blockId="PE12"/>
```

The parameter PE12 is a unique block identifier.

The last type of JSP-block in Vitrage is **Composites**. These blocks manage collections of other blocks and apply some logic to coordinate the code generation between inner blocks. Composites provide inter-block layout automation; this part of the framework was developed with a view to creating lightweight portals usable outside portal platforms. In their present form, composites are used to manage HTML surrounding blocks, e.g., when several blocks share a single frame that should be rendered if at least one of them produces any content.

Building Compartmentalized Applications with Vitrage

Vitrage Framework can help save resources and improve the process of developing Web applications.

Use

When assembling a JSP from blocks, the most common syntax is to use a custom tag from the blocks library provided

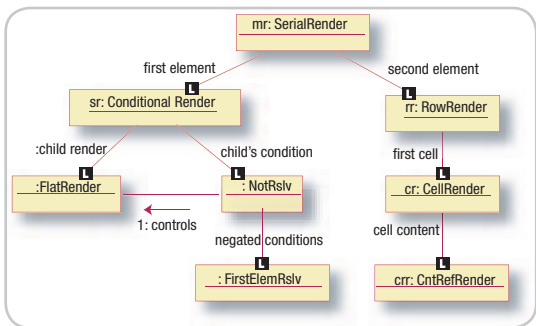


Figure 6 Composition of a block render

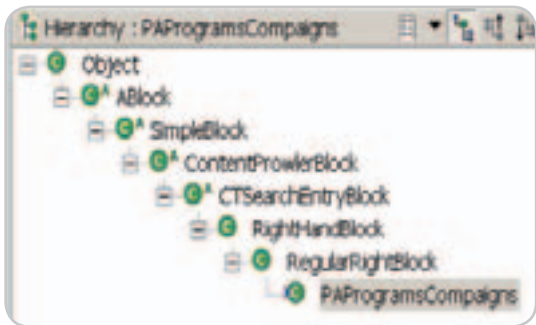


Figure 7 Implementation of PAMProgramsCompaigns

Nothing beats our racks

Absolutely nothing



CARRIER CLASS DATA CENTERS

- Highly Secure Guarded Facility
- 24/7/365 Network Operations Center
- 24/7/365 Technical Support
- Redundant Conditioned Air Systems
- Redundant Fiber Entry Points
- Multiple Uninterruptible UPS Systems
- Multiple 1250 KW Generators Onsite
- VESDA Early Warning Smoke Detection

Robert Marsh, Head Surfer

START YOUR OWN WEB HOSTING BUSINESS TODAY!

from
\$299*
Instant Activation!

Dedicated Server

- Dual Xeon 2.4 GHz
- 2 GB RAM • 2 x 73 GB SCSI HD
- Remote Console • Remote Reboot
- 2000 GB Monthly Transfer Included

Over 20,000 Servers!

1-800-504-SURF | ev1servers.net

**PLESK7
RELOADED**
Preferred Control Panel

IP Compliant. Price subject to change. Quantities Limited.
*Per month. Set-Up fees apply. See web site for complete details.

with the framework:

```
<blocks:useBlock blockId="PE1"/>
```

In this example, PE1 is a unique block ID that identifies a compartment in the page design. Alternatively, a JSP-block can be invoked directly from a scriptlet:

```
<% BlockFacade.useBlock("PE1", request) %>
```

Runtime code behind these constructs is not exactly the same. Custom tag style invocation has a small overhead: an object of class `InvokeBlockTag` that implements custom tag `blocks:useBlock` will be created for each invocation of each JSP page. With a scriptlet invocation, such overhead is eliminated. In either case, the HTML developer uses identical code, varying just the block ID.

Vitrage Container

Vitrage includes a *containier* that manages the lifecycle of JSP-blocks, including block initialization, caching and invocation, similar to the way the JSR 168 Portlet Container manages portlets. This similarity runs further: in Vitrage, only one instance of each JSP-block is created for the entire application. All JSP invocations reuse that single instance without the unnecessary creation and garbage collection overhead. JSP-blocks are thread-safe, and one instance of each block is created during application initialization and kept alive in the block cache. In some cases, that could save a significant amount of resources. By contrast, custom tags aren't thread-safe by design, and a new instance of each tag is created on each JSP invocation.

Resources & Prototypes

One of the selling points of tag library technology is the separation between content and layout. Normally, JSP content comes from a database and is rendered by custom tags, while the layout is specified directly in the JSP. This prevents content from affecting layout, as is often required by the presentation logic.

Since this approach doesn't work for compartmentalized applications, Vitrage has to use another way to support such separation. On the page level outside of the block scope, static layout elements are still kept in the JSPs. A compartment may contain both static and dynamic internal layout elements. The dynamic elements are generated in JSP-blocks, while static elements are located in resource bundles and loaded at application startup. This also lets the Vitrage Framework support JSP-block internationalization. While similar to how internationalization is provided in Struts, it could be more runtime effective; the only instance of a block is created on application initialization, while with custom tags, a resource will be read on each JSP invocation.

Development Tools

Vitrage provides a structured means for integration between compartments realized by JSP-blocks and the application's back-end, which relies on strongly typed universal transport objects such as `IblockData`, `Contentlet` and `ContentReference`.

Taking advantage of this architecture, Vitrage supports two modes for serving content to JSP-blocks. In the *production mode*, each block obtains its content from the request, where it was put by the controller. In the *test mode*, all invocations of `getData()` are intercepted and forwarded to `TestDataProvider` component, which supplies test data in the format expected by the block. It takes some work to fill `TestDataProvider` with test data, but lets front-end development move forward independent of the application back-end. This way, Vitrage supports testing and integrating front-end components on several levels:

- As individual JSP-blocks with JUnit test suite, with output redirected to an HTML file, which can be viewed in browser; this method doesn't require any application or Web server or any application back-end
- As complete JSP pages inside the

Web container without any Model or Controller components

- As a complete MVC application, bypassing the lack of test data in the database
- Conventional end-to-end integration testing.

Conclusion

Modern Web applications use increasingly complex pages with compartments and exhibit sophisticated presentation logic. Tag libraries, JSP templates and portlets don't give developers adequate support for effectively implementing such dynamic front-ends. Being highly compatible with various J2EE platforms, MVC frameworks and tag libraries, the Vitrage Framework offers extensive support for compartmentalized applications. In many cases it also improves support for the application development process and provides significant performance gains for the resulting application. In some cases it permits the development of applications that wouldn't be feasible with tag libraries, templates or portlets. Additionally, developing applications with the Vitrage Framework can economize resources and time. ☺

References

- *Vitrage Whitepaper*. <http://www.roundarch.com/about.html>
- *JSTL specification*. <http://jcp.org/aboutJava/communityprocess/final/jsr052/>
- *BEA Tag Library*. <http://e-docs.bea.com/workshop/docs81/doc/en/workshop/reference/tags/navJspTags.html>
- *More on Tag limitations*. <http://www.theserverside.com/articles/article.jsp?l=BestBothWorlds>
- *Separation of Business Logic from Presentation Logic in Web Applications* <http://www.paragoncorporation.com/ArticleDetail.aspx?ArticleID=21>
- *Tiles 101/201 by Patrick Peak* <http://www.theserverside.com/resources/article.jsp?l=Tiles101>
- *A Brief History of Tags* by Rich Rosen, JDJ, 2003, Volume:8, Issue:6 pages 10-22.
- *Enterprise BluePrints* <http://java.sun.com/blueprints/enterprise/index.html>

“One of the selling points of tag library technology is the separation between content and layout”

**IMPORT PEERS.*;
IMPORT EXPERTS.*;
IMPORT YOU.*;**

**// EXPORT DEADLOCK
// GO TO SDN.SAP.COM**



THE BEST-RUN BUSINESSES RUN SAP



You're stuck. You need answers. Maybe you have a solution to share with other SAP developers or a question for an SAP insider. Get the experts, partners and your colleagues to weigh in. Now there's a single collaborative destination where you can all converge: SAP® Developer Network. Nowhere else can you join a spirited discussion forum, download a trial of the latest SAP Web Application Server, take an e-learning course, and, in general, keep us on our toes.

// JOIN IN AT SDN.SAP.COM

if (wantToGoHomeSooner)



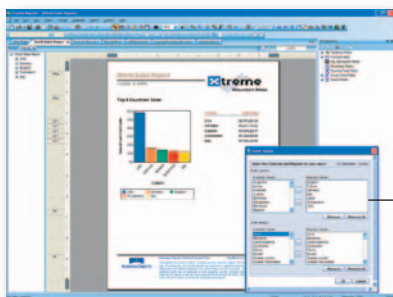
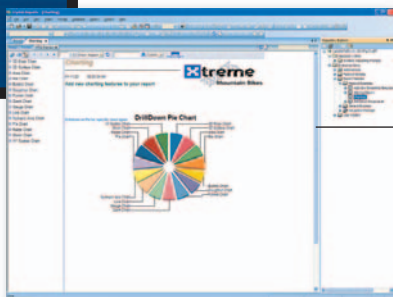
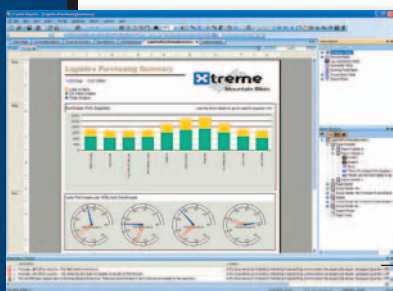
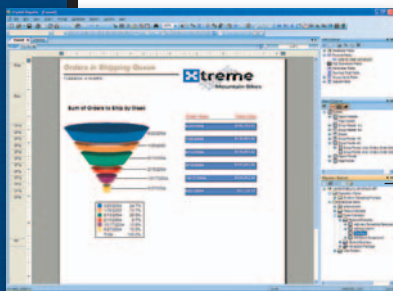
NEW Crystal Reports XI helps you spend less time integrating reports.

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.

Crystal Reports XI offers new productivity features and open deployment options to help you go home sooner.



Speed Data Access and Formatting

Visual Report Designer – Use an updated point-and-click designer to create reports and alleviate intensive coding.

Repository Explorer* – Increase your report design productivity by reusing existing report objects across multiple projects.

NEW Report Dependency Checker – Improve QA. Quickly find broken links, formula errors and dependency issues.

NEW HTML Preview Pane* – Quickly check web report design layout in an HTML preview pane.

Simplify Report Integration

Report Creation API* – Embed report creation functionality into your apps with no additional licensing fees, now included with the Developer edition.

NEW Cross-Platform API's* – Include server-side printing and sub-report configuration functionality.

Ease Report Delivery

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

NEW Dynamic Cascading Prompts – Minimize report maintenance with automatically updated pick lists and cascading prompts.

NEW Crystal Reports Server – Publish reports to the web for secure viewing, printing and exporting with a new report server option.

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.

*This feature is available as part of the Crystal Reports Server, included with Crystal Reports Developer edition.

Java Naming Services Internals

by Kishore Kumar

Implementing a simple client/server-based JNDI naming service

The Java Naming and Directory Interface (JNDI) is a standard API to access different naming and directory service implementations like LDAP. A naming service provides naming functionality and a directory service provides applications with directory functionality. The Java naming service is a fundamental component of every J2EE system.

JNDI consists of a client API and a service provider interface (SPI). The client application uses the client API to access various naming and directory services. The SPI lets naming and directory service implementations be plugged into the JNDI framework.

In this article, we will explore how to use the JNDI SPI to implement a simple client/server-based naming service.

Naming Service Architecture

Figure 1 shows the architecture and the various components of a typical Java naming service:

The first thing any JNDI client does is to create an *InitialContext* object. The *InitialContext* uses an *URLContextFactory* to create a *Context* implementation. The naming service provider supplies the *URLContextFactory* and the corresponding *Context* implementation. The JNDI *InitialContext* object delegates every naming method to the provider's context implementation.



Kishore Kumar works as a Java architect at U.S. Technology (www.ustr.com). He specializes in Java and J2EE applications.

Kishore_kumar@usswi.com

```
Hashtable properties=new Hashtable();
properties.put(Context.INITIAL_CONTEXT_
FACTORY,FACTORY_CLS_NAME);
InitialContext ic=new InitialContext(prop
erties);
Object appConfig=ic.lookup("java:comp/env/
appConfig");
```

Implementing the Java Naming Service – Server

Data Structure

A naming context refers to a set of name-object mappings. In a hierarchical naming system, a name in a context can map to a child context. At the server, the name-object mappings (*bindings*) can be stored in a *Map* data structure:

```
public class Store {
    Map bindings=new HashMap();
    public Binding get(String name) { // get
value from data }
    public void set(String name, Binding b) {
// put value into data }
}
```

A binding instance contains the triple: the name, object to be stored, and the object class name.

Naming Server Interface

The naming server is a remote interface that defines server methods that correspond to every method on the *javax.naming.Context* interface:

```
public interface NamingServer extends
```

```
Remote, Serializable
{
    public void bind(Name name, Object object,
String className) throws NamingException,Re
moteException;
    public Object lookup(Name name) throws Nami
ngException,RemoteException;
    // Other methods corresponding to methods
in javax.naming.Context
}
```

Name

Each naming method in the *Context* class has two overloaded forms – one that takes in a *String* name and another that takes in a *Name* object. For instance, the *Context* class has a method *lookup(String name)* and a method *lookup(Name name)*. The *Name* object is a collection of various sub-components in a name.

In the case of a name '*comp/env/appOwner*,' the *Name* object represents the list '*{comp,env,appOwner}*.' It's the *NameParser* of a naming system that parses the string representation of a name and creates the corresponding *Name* object. The parsing rules depend on the naming system and its naming parser implementation.

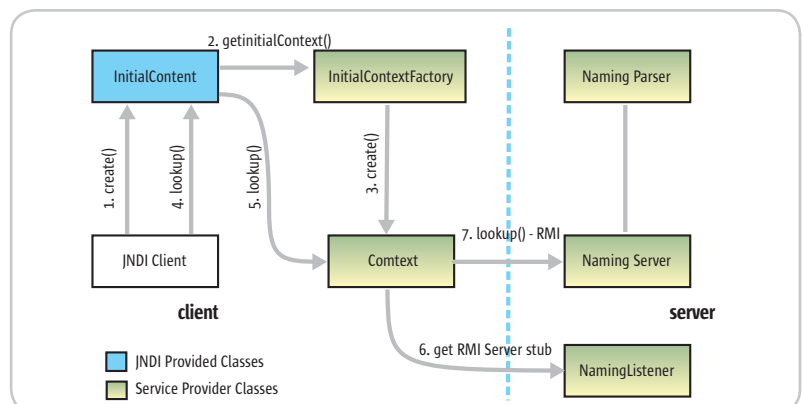


Figure 1 Java naming service components

There are two different kinds of structured names that implement the *Name* interface— *CompositeName* and *CompoundName*. A *CompositeName* represents a name that can span multiple naming systems.

For instance, the name ‘*cn=aperson,ou=ust_india,o=ust/profile/publications*’ represents two parts: an LDAP name ‘*cn=aperson,ou=ust_india,o=ust*’ and a File System name ‘*profile/publications*.’ When this name is used in a lookup operation, the method resolves through the LDAP naming service and passes into the File System naming service to find the target object.

In a *CompositeName* the various components are separated by a forward slash (‘/’) character. Hence, the given name has three components ‘{(cn=aperson,ou=ust_india,o=ust),(profile),(publications)}’. The first component represents a name (compound name) in the LDAP naming system and the last two components represent a name in the File System naming system. The mechanism by which a JNDI system resolves through multiple naming systems is known as a naming federation. We will explore this technique later in this article.

A *CompoundName* is a name in a single naming system. It is a convenience class that can parse a string representation of a name to create a *Name* object. Its parser can be customized by providing the required parsing syntax:

```
Properties syntax=new Properties();
Syntax.setProperty("jndi.syntax.
direction","left_to_right");
Syntax.setProperty("jndi.syntax.
ignorecase","false");
Syntax.setProperty("jndi.syntax.separa-
tor","/");
CompoundName name=new CompoundName("comp/
env/appOwner",syntax);
```

This parses the string representation using the syntax provided to create an equivalent *CompoundName* in the current naming system. A typical naming parser can also use the *CompoundName* to parse a string name to a *Name* object:

```
Public class NamingParser implements
NameParser{
    Public Name parse(String name) throws
NamingException {
        return new CompoundName(name,syntax);
    }
}
```

Naming Server Implementation

The naming server implementation uses the ‘*Store*’ data structure to store name-object mappings:

```
public class NamingServerImpl implements
NamingServer{
    private NameParser nameParser=new
NamingParser();
    private Store rootContextStore=new
Store();
    public void bind(Name name, Object
object,String className) throws
NamingException,RemoteException{
        Store contextStore= rootContextStore;
        Object[] storeNamePair=new Object[]{rootC
ontextStore,name};
        iterateNameComponents(storeNamePair);
        contextStore=storeNamePair[0];
        name=storeNamePair[1];
        // bind the name to the given object
        contextStore.put(name,new Binding(name,c1
assName,obj,true));
        // check for NameAlreadyBoundException
    }
    private void iterateNameComponents(Object[
] storeNamePair) {
        Store contextStore=storeNamePair[0];
        Name name= storeNamePair[1];
        // iterate the name till the last compo-
nent
        while (name.size() > 1){
            Object obj=contextStore.get(name.
get(0));
            // name.get(0) gets the first component
in the given name
            // obj should be a context
            If !(obj instanceof Store){ // throw
NotContextException }
            contextStore=(Store)obj;
            // resolve the remaining name in new
context
            // get remaining name component
            name=name.getSuffix(1);
        }
        storeNamePair[0]=contextStore;
        storeNamePair[1]=name;
    }
    // other methods
}
```

The *lookup()* method is similar to the *bind()* method:

```
public Object lookup(Name name) throws Nami
ngException,RemoteException {
```

```
Store contextStore= rootContextStore;
Object[] storeNamePair=new Object[]{contex
tStore,name};
iterateNameComponents (storeNamePair);
contextStore=storeNamePair[0];
name=storeNamePair[1];
// lookup the name in the resolved context
(store)
Binding binding=contextStore.get(name);
// check for NameNotFoundException
Object obj=binding.getObject();
If (obj instanceof Store) return name; //
to represent a context
else return obj;
}
```

The returned object can be a sub-context or a target object. If the object found represents a context (if it’s an instance of *Store*) then return the name itself to tell the client that the name represents a context. Otherwise, return the object found.

The *createSubContext()* method creates a sub-context (represented as a *Store* instance) in the current context. It returns a *Name* instance to indicate the new context:

```
public Name createSubContext(Name name)
throws NamingException,RemoteException{
bind(name,new Store()); // simplified
implementation
return name;
}
```

This will create a new context (represented by a store object) bound against the given name.

Naming Server Listener

A naming server listener listens for client connections from the *Context* (client) implementation and returns a naming server stub. The *Context* implementation uses this RMI server stub to invoke naming methods on the server:

```
NamingServerImpl server=new
NamingServerImpl();
Remote stub=UnicastRemoteObject.
export(server, rmiPort, rmiClientSocketFac-
tory, rmiServerSocketFactory)
MarshaledObject serverStub=new
MarshaledObject(stub);
ServerSocket serverSocket=new
ServerSocket(bindPort, backlog, bindAd-
dress);
// start a thread to wait for client con-
```



DESKTOP



CORE



ENTERPRISE



HOME

```

nections
Socket socket=serverSocket.accept();
// create a new thread and start it to listen for new connections
ObjectOutputStream oos=new ObjectOutputStream(socket.getOutputStream());
out.writeObject(serverStub);

```

Implementing the Java Naming Service - Client

Any JNDI service provider should provide an implementation for the *javax.naming.Context* interface and should provide an *InitialContextFactory* implementation that can create a new *Context* in the new naming system.

Context Implementation

This *Context* implementation is an RMI client and uses the server stub to invoke naming server operations.

A *Context* has overloaded forms of the same naming method: one taking a *Name* argument and the other taking a *String* name argument. The method that takes in a *String* name argument converts the *String* representation of the name to a *Name* object using a *NameParser* and invokes the other overloaded method:

```

public Object lookup(String name) throws NamingException {
    return lookup(getNameParser().parse(name));
}

```

References and Referenceable

A naming system can store serialized Java objects directly in its object store. An alternative and efficient mechanism is to use references. A reference can be used when the serialized representation is too big or the object can't be stored directly. An object is stored with an associated reference in the directory indirectly by storing its reference.

A reference is represented by the *Reference* class. A reference consists of an ordered list of addresses and class information about the object being referenced. Each address is represented by a sub-class of *RefAddr* and contains information on how to construct the object.

When the naming system stores an object and it implements the *Referenceable* interface, its *getReference()* method will be invoked to obtain a *Reference* object and stores the reference.

And when the naming system retrieves the object from its store and it's a *Reference*, it uses the information (object factory) on how to create the object to create the actual object and returns it.

State Factories and the bind() Method

A state factory transforms an object into another object (state).

A naming system implementation uses state factories to convert objects to an acceptable form so it can store them.

A state factory implements the *StateFactory* interface:

```

public Object getStateToBind(Object obj,Name name,Context ctx,Hashtable env);

```

The JNDI framework has support for state factories that naming service providers can make use of. The *NamingManager.getStateToBind()* method traverses the list of state factories specified in the *Context.STATE_FACTORIES* environment property and tries to find a factory that yields a non-null result. If no state factories yield a non-null result, the naming manager returns the given object itself.

The context *bind()* method uses state factories to convert the object to be bound to a state representation that's suitable for the naming system to store that object. Then it checks for the *Referenceable* and *Reference* properties of the object before storing the object:

```

public void bind(Name name,Object object) throws NamingException{
    Object state=NamingManager.getStateToBind(obj,name,this,env);
    If (state instanceof Referenceable) {
        state=((Referenceable)state).getReference();
    }
    if (state instanceof Reference) {
        className=((Reference)state).getClassName();
    } else { className=state.getClass().getName();}
    Name aName=getAbsoluteName(name);
    serverStub.bind(aName,object, className);
    // need to catch exception
}

```

Every context saves the environment used to create that context. Any child context created using the parent context will also inherit the environment (referred to as *env* in the code above).

Here, every name given to the context is assumed to be a relative name. However, the context implementation can be modified to handle names given as URLs (*java:comp/env/jdbc/aDataSource*).

The *getAbsoluteName()* method is a utility method that converts the relative name to an absolute name. For instance, if the context represents the name '*comp/env/jdbc/aDataSource*' then the fully qualified name is '*comp/env/jdbc/aDataSource*'.

Object Factories and the lookup() Method

An object factory produces objects. It accepts some information about how to create an object, such as a reference or a state, and then returns an instance of the actual object.

An object factory implements the *ObjectFactory* interface:

```

public Object getObjectInstance(Object info,Name name,Context nameCtx, Hashtable env)throws Exception;

```

A naming service provider uses the *NamingManager.getObjectInstance()* method to fetch objects before they are returned to the caller. This method traverses a list of object factories specified in the *Context.OBJECT_FACTORIES* environment properties and tries to find a factory that yields a non-null result. If the object is a reference then this method uses the object factory class named in the reference:

```

public Object lookup(Name name) throws NamingException {
    Name aName=getAbsoluteName(name);
    Object object=serverStub.lookup(aName);
    If (object instanceof MarshalledObject){
        object=((MarshalledObject)object).get();
    } else if (object instanceof Name) { // represents a context
        object=new ContextImpl((Name)object,serverStub,env);
    } else {
        object=NamingManager.getObjectInstance(object,name,this,env);
    }
    return object; // need to handle exceptions
}

```

You didn't compromise on your infrastructure... show your applications the same respect.



RICH WEB APPS

Need desktop interactivity within a web browser?



LIVE DATA

Wish your browser-based apps had Excel-like functionality?



UNIVERSAL DEPLOYMENT

Imagine building an application once, deploying it anywhere?



INDUSTRIAL STRENGTH

Want enterprise-scale Web apps that are reliable, secure and support 1,000's of concurrent users?

Nexaweb's rich client platform makes your apps available anytime, anywhere.

We know you're tired of being asked to do the impossible. We can help.

Nexaweb lets you leverage all the work you've done in client/server and deploy your apps on the Internet, without compromise.



“ Nexaweb was able to help us deliver significant cost savings while enhancing overall system performance over infrastructure alternatives. ”

Mike Kistner, CIO and
Vice President, Best Western



DESKTOP



CORE



ENTERPRISE



HOME

Initial Context Factory Implementation

When an `InitialContext` object is created, JNDI uses the `Context.INITIAL_CONTEXT_FACTORY` environment property to identify the Initial Context Factory implementation class and creates a context that represents the root of the naming system. The `InitialContext` uses this context to execute naming operations on that naming system:

```
public class InitialContextFactoryImpl
implements InitialContextFactory {
    public Context getInitialContext(Hashtable
env) throws NamingException {
        String providerURL=(String)env.
get(Context.PROVIDER_URL);
        Name name=getContextName(providerURL);
        serverStub=createServerStub(providerURL);
        return new ContextImpl(name,serverStub,en
v);
    }
}
```

The `getContextName()` is an utility method that parses the provider URL and returns the root name. If the provider URL isn't provided then the method returns an empty `Name` object to denote the root of the naming system. For instance, if the URL is `'protocol://serverName:port/comp/env'`, the `getContextName()` returns the context name `'comp/env'`.

Environment Properties

The environment properties can be provided when an `InitialContext` is created:

```
Hashtable env=new Hashtable();
env.put(Context.INITIAL_CONTEXT_
FACTORY,FACTORY_CLASS_NAME);
env.put(Context.PROVIDER_URL,"localhost");
// naming server URL
InitialContext context=new
InitialContext(env);
```

The data in the `Hashtable` are called environment properties. You can also set the state factories, object factories, URL context factories and several other standard properties as environment property values.

To simplify the task of setting up the environment required by a JNDI application, application resource files should be distributed along with application and service providers. An application resource file has the name 'jndi.properties' that lists the environment properties. The JNDI automatically reads all the resource files (`jndi.properties`) from the application's classpath and `JAVA_HOME/lib/jndi.properties`. JNDI environment properties can also be specified through system properties made available to the Java program via the `-D` command line option in the Java interpreter. If the environment parameter is specified using more than one source, then the search order is:

1. `InitialContext` constructor environment parameter
2. System Property
3. Application resource files (`jndi.properties`) in system classpath

For properties like `INITIAL_CONTEXT_FACTORY`, the first value found is used and for properties like `OBJECT_FACTORIES`, all the values found are concatenated into a single colon-separated list.

URL Context Factory Implementation

URL strings can be used as names to the `InitialContext`. A URL consists of the form `scheme:scheme-specific-parts (java://localhost:8080/comp/env)`. Here, 'java' is the scheme and the rest of the string forms the scheme-specific-parts.

In JNDI, every name is resolved relative to a context. Every name supplied to a context is a relative name. When an URL is used as a name to the `InitialContext`, it represents an absolute name. The `InitialContext` class diverts the method invocation so that it's processed by the corresponding URL context implementation rather than any underlying initial context implementation.

When the `InitialContext` receives an URL as a name argument to one of its methods, it looks for a URL context implementation. It uses the `Context.URL_PKG_PREFIXES` environment property. This property contains

a list of package prefixes. Each item in the list refers to an URL context factory. The factory name is constructed using the following rule:

```
package_prefix.scheme.schemeURL-
ContextFactory.
```

The package prefix `'com.sun.jndi.url'` is always appended to the list of URL context factory package prefix. Now suppose that the `URL_PKG_PREFIXES` property contains `'com.widget.com.wiz.jndi'` and the URL name is `'ldap://localhost:389'`, JNDI will search for the following URLContextFactories:

```
com.widget.ldap.ldapURLContextFactory
com.wiz.jndi.ldap.ldapURLContextFac-
torycom.sun.jndi.url.ldap.ldapURLCon-
textFactory
```

`InitialContext` will try to instantiate each class in turn and invoke the `getObjectInstance()` (object factory method) until one of them produces a non-null result.

Link References

A link reference is a symbolic reference to an object in the naming system. Suppose there is following name in the `InitialContext`: `some/where/over/there`.

You can create a link reference to `'some/where'` and bind it to the name `'here'`. Subsequently using the name `'here/over/there'` is the same as using the name `'some/where/over/there'` in the naming methods.

A link reference is represented by a `LinkRef` instance. A link reference can be de-referenced using the `context.lookupLink()` method.

A `LinkRef` object can be resolved as follows:

```
String ref=linkRef.getLinkName();
If (ref.startsWith("./")) {
// Treat it as a relative name to the cur-
rent context
linkResult=lookup(ref.substring(2));
}
else { // Treat it as an absolute name
```



The Java naming service is a fundamental component of every J2EE system”


```
linkResult=new InitialContext(env).lookup(ref);
}
```

The naming methods in the context class dereference a link automatically. For instance, the `lookup()` method dereferences the link if the object returned is an instance of `LinkRef`:

```
Object result=serverStub.lookup(name);
If(result instanceof LinkRef) {
    result=resolveLink(result);
}
```

Naming System Federation

Federation is the process by which a composite name (name that spans different naming systems) is resolved through the different underlying naming systems. The whole process is transparent to the user who simply provides a name to the naming system; the naming system takes care of the switching to other naming systems.

The composite name `'cn=aperson,ou=ust_india,o=ust/profile/publications'` has three components:

```
cn=aperson,ou=ust_india,o=ust
profile
publications
```

In a composite name the components are separated by the `'/'` character. The first component is resolved in the LDAP naming system and the second and third components are resolved in the File System naming system.

The `lookup()` operation begins on the LDAP context and the name component `'cn=aperson,ou=ust_india,o=ust'` resolves to a next naming system (nns) reference. The naming server identifies this and throws a `CannotProceedException` (from the `iterateNameComponents()` method)

```
If (obj instanceof Reference) {
If (((Reference)obj).get("nns")!=null) {
    cpe.setResolvedObj(obj);
    cpe.setRemainingName(name);
    throw cpe;
}
}
```

The `'nns'` pointer is an URL that points to the next naming system (the File System in this example). The context will handle and process the `CannotProceedException` as shown below:

```
try {
    result=serverStub.lookup(name);
```

```
} catch (CannotProceedException cpe) {
Context nnsCtx = NamingManager.getContinuationContext (cpe);
result=nnsCtx.lookup(cpe.getRemainingName());
}
```

The `NamingManager's getContinuationContext()` method uses the configured object factories to create the next naming system context. Since the `'nns'` pointer is a URL, it uses the URL Context Factories to create the new context. The same operation is continued on the remaining name left to be resolved on this new context.

Summary

This article explored the concepts of implementing a simple client/server-based JNDI naming service. It examined the various basic concepts of a naming server like Context, `InitialContextFactory` and `Reference`. It also examined advanced concepts like `URLContextFactory`, `StateFactory`, `ObjectFactory`, Link References and Naming Federation. ☺

Reference

JNDI Tutorial: <http://java.sun.com/products/jndi/tutorial/>



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.





Calvin Austin

Core and Internals Editor

Ten Years of Java Technology

This year will mark the tenth anniversary of the official launch of Java technology. It seems like only yesterday. No doubt there will be celebrations similar to the five-year anniversary, so I thought I would take this opportunity to step back in time and track Java's course.

In January 1996, less than a year before that first launch, the first full developer kit, JDK 1.0.2, was released. This was my first experience of the Java platform. Like many other developers I had been using C and C++ and myriad third-party libraries. Suddenly the ease with which anyone could build a UI, or a web applet, and make an application both thread- and networking-aware was exciting. I attended the first JavaOne in 1996 and it captured that energy, even with only 6000 attendees it was a sellout. Sessions overflowed, handouts disappeared in minutes, and were never to re-appear at any JavaOne conference again; many speakers were overwhelmed by speaking to a large conference for the first time.

The JDK 1.1 release appeared a year later and bumped along by way of maintenance updates for many years, finally ending with 1.1.8.

Some of you may remember that the only browser that initially supported 1.1 was Sun's own Hotjava browser. This lag in support for the latest runtime would lead to the modular Java Plugin and Java Web Start technology.

JDK 1.1 also introduced JDBC, RMI and the JavaBean model. The JavaBean component model, while introducing the powerful getter/setter pattern to the Java platform, also introduced the infamously deprecated methods in AWT. To move to the JavaBean pattern

with as little risk as possible, AWT code that needed updating simply called the deprecated methods, which made removing them later unlikely.

AWT also introduced the event-delegation pattern that would be heavily used by another step on the Java roadmap, the Swing project.

Project Swing, or the JFC components, had a parallel release train before being integrated into the JDK. Anyone remember com.sun.swing? The Netscape browser team already had a technology called IFC that Netscape had acquired and this was used as the basis for JFC. JFC was a pure Java graphical toolset and required a little support from AWT. However, the amount of work required was huge. Essentially anyone who was working on AWT was moved to JFC and Swing. All new development for features like accessibility and full drag-and-drop were earmarked for Swing only. The next step was to merge the Swing code base into JDK 1.2.

JDK 1.2 was supposed to be called JDK 2.0. Since its release was close to the millennium, even Java 2000 was considered. The naming discussions resulted in Java 2 version 1.2. The release didn't just include Project Swing, but it did include the Collections API, a new Java 2D rendering engine and a new sound engine. The last two technologies were adapted from existing third-party products and their integration put a strain on the release process. Some of the bugs introduced by this integration weren't fixed until the 1.4 maintenance releases and J2SE 5.0

Most developers have probably forgotten 1.2.1. It was a short-lived security bug fix. The true maintenance bug fix

was 1.2.2. JDK 1.2.2 was also the first time Sun released a JVM port on Linux. The JVM itself was called the classic JVM and used a JIT compiler.

Waiting in the wings was the Hotspot JVM. Sun had acquired the technology that was used to power Smalltalk and had spent a lot of cycles getting it release-ready.

Unlike the JIT compiler, the Hotspot product was a full JVM in its own right. It used native operating system threads, where the classic JVM could also use the userspace threads called green threads and introduced new garbage allocation techniques, finer thread management and faster monitor locks.

J2SE 1.3 was released in 2000 and introduced the Hotspot JVM on all platforms. With such a fundamental change, it took until 1.3.1 for the JVM to be supported by all the tool interfaces.

The last five years are fresher in everyone's memory. J2SE 1.4 arrived in 2002, and introduced NIO, Java Web Start, a 64-bit JVM and Swing focus, performance tweaks and the logging API. It was followed by the 1.4.1 maintenance release, which previewed an Itanium port and new garbage collectors. J2SE 1.4.2 brought the 1.4 release train into the station.

This brings us to the present times with J2SE 5.0. J2SE 5.0 focuses on improved startup time, new language features and system monitoring and improved product quality.

The Java platform has certainly come a long way in 10 years, but I'm sure you'll agree it's been an interesting ride. ☺

Resources

- <http://java.sun.com/features/2000/06/time-line.html>

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

“When Java was born, it was suddenly easy to build a UI, or a web applet, or make an application thread- and networking-aware”



Do it yourself.

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



xPhoneApp™

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

Java Annotation Facility – A Primer

by Krishan Viswanth

JDK 5 has changed source code generation in a seminal way

The 5.0 release of JDK introduced a slew of new features. A powerful technique that resulted from the JSR-175 recommendation is the Program Annotation Facility. It can annotate code in a standard way and automate the generation of source code or configuration files, helping cut down on boilerplate code.

At the moment, the closest thing to annotating source and generating support file/code is through java doc tags. The popular ones are `@deprecated`, `@author`, `@param` etc. However, these tags are pretty static by nature and the information they define isn't encoded in the class file by the compiler so it's not available at runtime. A popular implementation of this concept is XDoclet. This is an Open Source utility that lets a developer add metadata or attributes to source as java doc tags. Appropriate source files or configurations, such as deployment descriptors, are generated later using the ANT task provided by XDoclet. (The source code can be downloaded from www.sys-con.com/java/sourcec.cfm.)

The core Java language has always had some form of ad hoc annotation scheme. Java doc tags are an example. Another example is the keyword `transient`, which is used to mark a member variable so it can be ignored by the serialization subsystem.

All this changed with the introduction of JDK 5.0, which adds a general-purpose customizable annotation mechanism.

This facility consists of syntax for declaring annotation types, syntax for annotating declarations, APIs for reading annotations, a class file representation for annotations and an annotation-processing tool.

Annotation and Annotation Types

The first step in the process is defining an annotation type. This is pretty simple to do and looks familiar as well. An annotation-type declaration looks like an interface declaration except an `@` symbol precedes the interface keyword. The method declaration that goes between the braces of this declaration defines the elements of the annotation

type. Of course, since we are annotating the code and not defining behavior, logically speaking, these methods shouldn't throw any exception. That means no throws clause. }
Another restriction is that the return type for these methods is restricted to primitives: String, Class, enums, annotations and arrays of the preceding types. The complete lists of restrictions are as follows:

- No extends clause is permitted. Annotation types automatically extend a marker interface, `java.lang.annotation.Annotation`.
- Methods must not have any parameters.
- Methods must not have any type parameters (in other words, generic methods are prohibited).
- Method return types are restricted to primitive types: String, Class, enum types, annotation types and arrays of the preceding types.
- No throws clause is permitted.
- Annotation types must not be parameterized.

The following code snippet defines an annotation type for a servlet. Presumably, we could use this definition to annotate a servlet and then have an annotation tool generate web.xml. Here we define no args methods that define the various XML attributes/elements found in web.xml. For conciseness we have left out elements like `init`, `load on startup`, `icon` etc.

```
public @interface Servlet {
    String servletName();
}
```

```
String servletClass();
String displayName();
String description();
}
```

Declaring Annotation

Now that we have the annotation-type defined we can annotate our servlet using the defined annotation type. Annotation is a new kind of modifier that contains an annotation type with zero or more member-value pairs. If a member has a default value defined in the annotation-type member declaration then the value can be omitted, otherwise, annotation must provide a member-value pair for all members defined in the annotation type. Annotation can be used for modifiers in any declaration – class, interface, constructor, method, field, enum, even local variable. It can also be used on a package declaration provided only one annotation is permitted for a given package. In our case we are annotating at the class level and the annotation precedes the access modifier public.

```
@Servlet(
    servletName="AnnotatedServlet",
    servletClass="com.jdj.article.servlet.
AnnotatedServlet",
    displayName="AnnotatedServlet",
    description="This is an example Annotated
Servlet"
)
public class AnnotatedServlet extends
HttpServlet{...}
```



Krishan Viswanth currently works for Capgemini, KS. He has over nine years of IT experience spanning a variety of technologies.

viswanath.krishnan@capgemini.com

Meta Annotation	Purpose
@Documented	Indicates that annotations with a type are to be documented by javadoc and similar tools by default.
@Inherited	Indicates that an annotation type is automatically inherited.
@Retention	Indicates how long annotations are to be retained with the annotated type. Three retention policies could be specified and they are CLASS, RUNTIME and SOURCE. If no retention annotation is present on an annotation-type declaration, the retention policy defaults to CLASS.
@Target	Indicates the kinds of program element an annotation type applies to. The Enum ElementType defines various annotation types and it's used with the Target meta-annotation type to specify where it's legal to use an annotation type

Table 1 Meta annotation

Annotation	Purpose
@Deprecated	A program element annotated @Deprecated is one that programmers are discouraged from using, typically because it's dangerous, or because a better alternative exists. Compilers warn when a deprecated program element is used or overridden in non-deprecated code.
@Override	Indicates that a method declaration is intended to override a method declaration in a superclass. If a method is annotated with this annotation type but does not override a superclass method, compilers are required to generate an error message.

TABLE 2 Annotation tags

Now, to tie all these together we will look at how to build a simple annotation-driven framework. However, before we start looking at concrete code samples, we will go over a bit more of the theory behind this important addition to the core language API.

Meta Annotation Types

The API provides some annotation types out-of-the-box that can be used to annotate the annotation types. These standard annotation types are also known as meta-annotation types. The details are provided in Table 1.

Standard Annotation

The Tiger release of the JDK also bundles a set of standard annotation types. Table 2 defines the annotation tags and their purpose.

Annotation Retention

The consumers of annotation fall into three categories.

- **Introspectors:** Programs that query runtime-visible annotations of their own program elements. These programs will load both annotated classes and annotation interfaces into the virtual machine.
- **Specific Tools:** Programs that query known annotation types of arbitrary external programs. Stub generators, for example, fall into this category. These programs will read annotated classes without loading them into the virtual machine, but will load annotation interfaces.
- **General Tools:** Programs that query arbitrary annotations of arbitrary external programs (such as compilers, documentation genera-

tors and class browsers). These programs will load neither annotated classes nor annotation interfaces into the virtual machine.

The grouping of annotation consumers mentioned above is determined by the retention policy that is specified by the RetentionPolicy enum present in the java.lang.annotation package. If the retention policy is 'CLASS' then the annotations are recorded in the class files but are not retained by the virtual machine. If the retention policy is 'RUNTIME' then the annotations are recorded in the class file and are retained by the VM at runtime. The value 'SOURCE' causes the compiler and VM to discard the annotation.

Annotation Processing Tool

The annotation processing tool (apt) found in JAVA_HOME/bin directory is a command-line utility that ships with JDK 5.0. This tool looks for annotation processors based on the annotation in the set of specified source files being examined. Essentially the annotation processor uses a set of reflective APIs and supporting infrastructure to process the annotations.

When invoked, the apt goes through the following sequence of operations: First, it determines what annotations are present in the source code being operated on. Next, it looks for annotation processor factories. It then asks the

Think grid computing is only for specialized applications? **Think again!**

xTier™ / GRID
Grid Computing for Java - Made Simple

Bring the power of GRID
to your Java/J2EE applications

Fitech Laboratories Inc.
sales: +1.646.495.5076 or sales@fitechlabs.com
support: support@fitechlabs.com - 24/7/365

www.fitechlabs.com/grid

factories what annotations they process and, if the factory processes an annotation present in source files being operated on, the apt asks the factory to provide an annotation processor. Next, the annotation processors are run. If the processors have generated new source files, the apt will repeat this process until no new source files are generated. This high-level sequence is indicated in Figure 1.

To write a factory class, a developer has to rely on packages that aren't part of the standard SDK. The packages used are:

- **com.sun.mirror.apt:** interfaces to interact with the tool.
- **com.sun.mirror.declaration:** interfaces to model the source code declarations of fields, methods, classes, etc.
- **com.sun.mirror.type:** interfaces to model types found in the source code.
- **com.sun.mirror.util:** various utilities for processing types and declarations, including visitors.

These packages are bundled in tools.jar, and so this jar file needs to be set in the classpath to write and compile the factory class. Assuming that the path and the classpath are set correctly, the annotation processing tool can be invoked from the command prompt by typing 'apt' followed by the tools command-line parameters.

Using Annotation for Generating Struts-config.xml

For a concrete understanding of the technology it's imperative that we dive into writing code and see for ourselves how it works. We will look at how to go about generating struts-config.xml by annotating code and using the apt tool. This article assumes that the reader is familiar with the Open Source MVC framework struts. We will look at how to generate the configuration details and the declarative programming semantics provided in the struts-config.xml by annotating the source code.

Before we get to the nitty-gritty of annotating and generating configuration files, we need to understand why this needs to be done.

We need to do this because a developer who has to write code using the struts framework finds himself copying and pasting information from the source code to the deployment descriptor and vice versa. For instance, if we change the name of the Action class and don't change

the XML file, the application doesn't work correctly. The way to sidestep this issue is to isolate the changes to one location and let the utility tool generate the deployment descriptor. We will cover how to use the metadata facility to achieve this automatic configuration file generation.

Before we start we have to make sure that we have the right development tools to do what we're trying to do. Currently very few IDEs support Java 5. Among Open Source IDEs, NetBeans 4.0 beta 2 looks promising (in spite of a few runtime exceptions) so most of the code in this article was written and tested with it. The first step in the process is defining the annotation types for the various elements that make up the struts framework. The component parts of the framework are: Action, Form bean (also known as Action Forms), Exceptions, Validator, Plug-ins etc.

The annotation type for Struts Action is as follows:

```
package com.jdj.article.atypes;
import java.lang.annotation.*;

@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.TYPE)
public @interface StrutsAction {
    String name();
    String path();
    String scope() default "session";
    String input() default "";
    String roles() default "";
    String validate() default "";
    String parameter() default "";
    StrutsActionForward[] forward();
    StrutsActionException strutsActionException();
}
```

As you can see, I have the annotation types as return types. This is necessary because we could have forward and exception elements defined by the action ele-

ment. The XML snippet from that struts-config.xml that we're trying to generate will clarify the need for embedding other annotation types in the action annotation-type declaration.

```
<action path="/SubmitLogon"
        type="example.LogonAction"
        name="LogonForm"
        scope="request"
        input="logon">
    <forward name="failure" path="/MainMenu.do"/>
    <forward name="success" path="/someValidPage.jsp"/>
    <exception key="expired.password"
              type="example.ExpiredPasswordException"
              path="/ExpiredPassword.do"/>
</action>
```

The next step is to annotate the code with the annotation type. In our case we will annotate a struts action class. This is done as follows:

```
package ...
import ...
@StrutsAction(
    name=" example.LogonAction",
    path="/SubmitLogon",
    forward = {@StrutsActionForward(
        name = "failure",
        path = "/MainMenu.do" }},
    strutsActionException =
        @StrutsActionException(
            key = "expired.password",
            type = "example.ExpiredPasswordException",
            path = "/ExpiredPassword.do"
        )
)
public class ExampleAction extends Action {
    /** Creates a new instance of
```

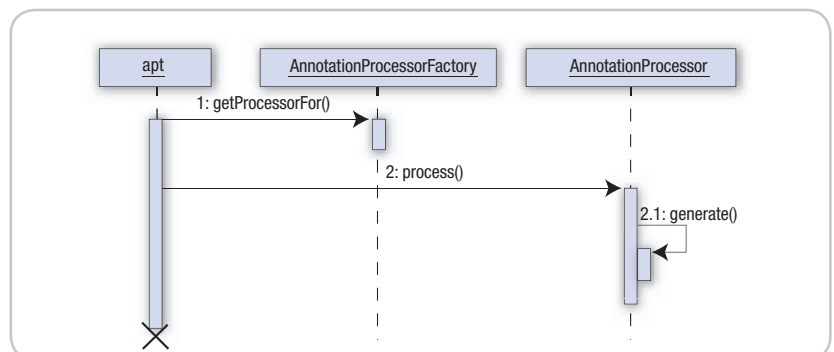


Figure 1 Annotation processing

```

ExampleAction */
public ExampleAction() {
}

public ActionForward execute(ActionMapping
mapping,
ActionForm form,
HttpServletRequest request,
HttpServletResponse response)
throws Exception {
return null;
}
}

```

Before tackling how to generate code or other dependent files such as deployment descriptor, we should clarify a couple of aspects presented in the code snippet above. The StrutsAction annotation type defines many elements but only a few of them were actually declared in the real annotation. This is perfectly valid because the elements that don't have a value declared will take the default value defined in the annotation type. In this case most of the values defaulted to an empty string but the scope element has a default value of session. So if the developer doesn't explicitly specify the value, the default value will be used. In the StrutsAction annotation type, we

also have the return type for the forward element as an array of StrutsActionForward elements. So annotation is done by separating the values by a comma wrapped in braces.

```

forward = {@StrutsActionForward( ... ),
@StrutsActionForward( ... ) },

```

In this code snippet two action forward element are specified and both of them are enclosed by a brace separated by a comma. Now that we've talked about annotation types and annotation, the core concepts of annotation have been pretty much covered.

Generating Code/Supporting Files

The final steps in the process are the procedures for generating code or support files like deployment descriptor. This could be the hardest part depending on the complexity of what we are trying to do. If we are generating source code based on annotation information then we need to have some kind of parser generator to generate the code. JavaCC hosted at www.java.net is an excellent parser generator implementation. Since we are generating an XML file, we used JDOM API to generate the deployment descriptor for struts.

There are two ways to do the code generation. The first is to write a driver code that uses the reflection API and the annotation API to figure out what annotations are present and then operate on them appropriately. The other is to use the annotation processing tool bundled with the JDK to process the annotation.

In the section above we explained how the annotation processing tool works. The annotation processor that's returned by the factory does the actual work of generating code. In the core processing we loop through the annotation types and have a visitor based on the Gang of Four's visitor design pattern do the work. This visitor processes the annotation on the class declaration, method declaration, package declarations etc. and does the file generation.

The code snippet for the factory, annotation processor and a simple visitor is as follows:

```

public class StrutsConfigGenerator implements
AnnotationProcessorFactory{
private static final Collection<String>
supportedAnnotations
= unmodifiableCollection(Arrays.asList(""));
private static final Collection<String>
supportedOptions =
emptySet();
}

```

Javavavoom!

Get a high-performance, transactional storage engine that's 100% Java.

Berkeley DB Java Edition

Download at www.sleepycat.com/bdbje

Now there's a high-performance storage engine that loves Java just as much as you do: Berkeley DB Java Edition (JE). Brought to you by the makers of the ubiquitous Berkeley DB, Berkeley DB JE has been written entirely in Java from the ground up and is tailor-made for today's demanding enterprise and service provider applications.



Berkeley DB JE has a unique architecture that's built for speed. The software executes in the JVM of your application, with no runtime data translation or mapping required. And because it supports J2EE standards such as JCA, JMX and JTA, you can be sure you have the widest range of options.

Experience the outstanding performance of Berkeley DB JE for yourself.

Download Berkeley DB Java Edition today at www.sleepycat.com/bdbje, and view the presentation "Design and Implementation of a Transaction Data Manager."



```

public StrutsConfigGenerator() {}

public Collection<String> supported
AnnotationTypes() {
    return supportedAnnotations;
}

public Collection<String> supported
Options() {
    return supportedOptions;
}

public AnnotationProcessor get
ProcessorFor(
    Set<AnnotationType
    Declaration> atds,
    AnnotationProcessor
    Environment env) {
    return (AnnotationProcessor)new
    WebGeneratedAp(env);
}

private static class WebGeneratedAp
implements
AnnotationProcessor {
    private final AnnotationProcessor
    Environment env;
    WebGeneratedAp(AnnotationProcessor
    Environment env) {}

    public void process() {
        for (TypeDeclaration typeDecl :
            env.getSpecifiedType
            Declarations()){
            // GOF visitor Design Pattern
            applied here!
            typeDecl.accept(get
            DeclarationScanner(
            new WebGeneratorClass
            Visitor(), NO_OP));
        }
    }

    private class WebGeneratorClassVisitor
    extends
SimpleDeclarationVisitor {
        public WebGeneratorClassVisitor(){
        }

        public void visitClassDeclaration(
            ClassDeclaration d)
        {
            // Do some meaningful work here!!!
        }

        // You could have more specific decla-
        rations here
    }
}
}

```

An important thing that we need to take note of is the `getProcessorFor` method in the factory. This method returns the annotation processor and, in our case, is an instance of the `WebGeneratedAp` class. This class extends `AnnotationProcessor` and implements the `no args process` method. Here we loop through all the type declarations and then have the type declaration accept our visitor. Our visitor is `WebGeneratorClassVisitor` and has methods to handle the areas where annotations occur like class declaration, method declaration and package declaration. It's here that we read the annotation, find out what the value is and operate on it accordingly. The complete working copy of the source is provided along with this article.

The final step in the process is to invoke the Annotation Processing Tool. This is typically done through the command line as follows:

```

apt -cp [Classpath] -nocompile
-factory com.jdj.article.gen.
StrutsConfigGenerator
-Astruts-config=F:\Article\Dev\Annotation\
generated\struts-config.xml <path>\
ExampleAction.java

```

In this method of invocation we used the `-nocompile` to ensure that the source file specified isn't compiled. The `-classpath` option specifies the classpath required by the processor. The `-A` option is ignored by the APT tool and is used by the annotation processor to get any specific processing-related information. In our case, we specified the location of the `struts-config.xml` in the file system.

The last step is to process the files with the annotation processor. Besides the command line, one could write an ANT Custom Task to generate the code. We strongly recommend that this be done. Since more and more projects are using ANT for automating the build process, it makes more sense to do an ANT build for automating code generation as well.

Writing an ANT task is easy and done by writing a custom task and extending the `org.apache.tools.ant`. Task class. Once that's done, the next step is to define the attributes and the corresponding setter methods for the attributes. The core processing logic is

provided in the execute method called by the ANT framework. In our case, we delegate the bulk of the code generation work to the `com.sun.tools.apt.main.Main` class from the `tools.jar`. This is the actual byte code that's also used by the `apt` command line.

The following XML snippet from an ANT build shows how use the custom ANT task:

```

<taskdef name="apt"
    classname="com.article.jdj.
    annotation.task.APTTask" >
<classpath>
<pathelement
    location="F:\Article\Dev\Annotation\
    dist\Annotation.jar"/>
</classpath>
</taskdef>
<target name="main">
<apt factory="com.jdj.article.gen.
StrutsConfigGenerator" src="F:\
Article\Dev\Annotation\src\com\jdj\article\
action\*
    .java"
    processorOption="struts-config=
    F: \Article\Dev\Annotation\generated\
    struts-config.xml" >
<classpath>
<pathelement location="C:\Sun\AppServer\
lib\j2ee.jar"/>
<pathelement location="...">
</classpath>
</apt>
</target>

```

Summary

In this article we introduced the idea of defining annotation types, how to use defined types as annotation and how to go about generating code or other supporting files based on annotation. This new and powerful technology is transforming the way we code. As more and more tools and products start using this facility, developers will realize improved quality in their code and a concomitant increase in productivity. ☺

Resources

- *JavaCC*: <https://javacc.dev.java.net/>
- *JDOM*: <http://www.jdom.org/>
- *APT*: <http://java.sun.com/j2se/1.5.0/docs/guide/apt/index.html>
- *Annotations*: <http://java.sun.com/j2se/1.5.0/docs/guide/language/annotations.html>
- *JSR 175*: <http://www.jcp.org/en/jsr/detail?id=175>

WebRenderer™



Standards compliant embeddable web browser component

WebRenderer is a cutting edge embeddable Java™ web content rendering component that provides Java applications and applets with a fast, standards compliant HTML and multimedia rendering engine. WebRenderer provides a feature-packed API including complete browser control, a full array of events, JavaScript interface, DOM access, document history and more.

Why WebRenderer?

- Standards Support (HTML 4.01, CSS 1 & 2, SSL, JavaScript, XSL, XSLT etc.)
- Exceptionally Fast Rendering
- Predictable Rendering
- Scalability (deploy in Applications or Applets)
- Security (based on industry standard components)
- Stability and Robustness

Embed WebRenderer to provide your Java™ application with standards compliant web content rendering support.

To download a 30 day trial of WebRenderer visit
www.webrenderer.com

JadeLiquid™
Software

Copyright JadeLiquid Software 2004. JadeLiquid and WebRenderer are trademarks of JadeLiquid Software in the United States and other countries. Sun, Sun Microsystems, the Sun Logo and Java are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries. All other trademarks and product names are property of their respective owners.

Using JDBC & the Template Method Pattern for Database Access

Getting there with a minimum of JDBC knowledge

by Keith Reilly

JDBC is a simple and flexible way to access a relational database. The knock on JDBC is that it forces a developer to familiarize himself with its API, whose use can often result in reams of duplicate or similar code: get a connection, execute a statement, parse a result set, etc. All of which needs to be wrapped in a try-catch block and synchronized.

However, in this article, I'll describe how to use the template method pattern to centralize an application's JDBC code into a single class that can be extended using a minimum of JDBC knowledge.

Template Method Pattern

The template method pattern is a way of expressing a single task as a series of smaller tasks, represented as methods on a base class. These methods can then be overridden in sub-classes to redefine parts of the overall behavior.

Database access can be broken into steps. For writing to the database, the steps are: get a connection, create a statement and execute the statement. Reading from the database requires the additional step of parsing the result set. And all of this must happen within the context of a try-catch block and a transaction.

And all this lends itself to the template method pattern.

Database access can be simplified by defining the steps needed to execute a SQL statement in a single base class. Some of these methods can be concrete and some left to be defined in the sub-class. A developer can then create sub-classes for each of the different SQL statements he plans to execute and override only those steps necessary.

The Solution

Start with an abstract base class that contains all the steps necessary for executing either a read or a write to the database using JDBC. I call my class BaseStatement (shown in class diagram 1).

The method execute() contains the steps taken to execute a statement (see Listing 1).

Get the Connection

The first step is to get a Connection to the database. This is ordinarily done from a JDBC DataSource. Listing 2 shows BaseStatement's getConnection() method.

On line 4, the method checks to see if the statement already has a Connection, and if so, it returns it. (Why the statement might already have a Connection is discussed below in the section entitled Managing Transactions) If not, on lines 7-9, the method gets a reference to the DataSource from the JNDI directory, retrieves a Connection from the DataSource pool and stores it in the member variable called connection. Before returning the Connection to the calling method, getConnection() turns off autocommit (line 12).

Prepare the Statement

The next step is to get the statement. I've chosen to use a PreparedStatement. Listing 3 shows the getPreparedStatement method as well as the following step of processing the statement (inserting the data into the SQL statement).

On line 4, the getPreparedStatement method creates a PreparedStatement from the Connection by passing to it the preparedStatementText. The preparedStatementText was passed to the BaseStatement object through the constructor and held in the member variable called preparedStatementText. The values to be inserted in the prepared statement are also passed to the Statement object via the constructor in the form of a List and stored in the member variable called arguments (see Listing 4).

Listing 4 Both the SQL text and the list of arguments to be inserted into the Statement are passed to the constructor and stored in member variables.



Keith Reilly is a Java architect with MFS Investment Management in Boston.

kreilly@mfs.com

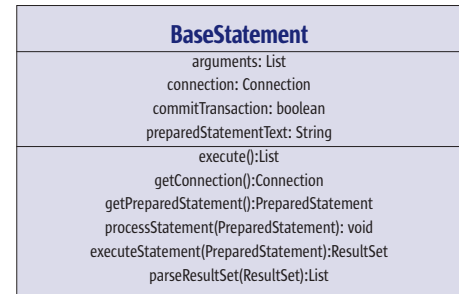


Table 1 Class Diagram

The processStatement method (line 7 in Listing 3) assumes that the first object in the variable, arguments, is the first value to be inserted into the PreparedStatement. The method that constructs the Statement must place the arguments in the List in the appropriate order.

This works equally well if there are no arguments. The for statement on line 11 (Listing 3) will fail immediately and no values will be inserted into the statement. This is exactly what we want when executing statements that require no processing (such as SELECT * FROM COMPANIES_TABLE).

Executing a Statement

The next step in the template is executeStatement, which will execute the statement constructed in the previous steps and return the result set (if any). It is defined as abstract in the BaseStatement class because statements that modify the database differ from query statements. Query statements are executed through the PreparedStatement's executeQuery method, which returns a ResultSet. Modifying statements call the method execute(), which returns a boolean. I'll use sub-classes to distinguish between statements that query and those that modify. These sub-classes will define the appropriate executeStatement behavior.

Querying the database

For querying the database, I've created a sub-class called SelectStatement that defines the method executeStatement() to return the ResultSet returned from a call to the method executeQuery off of the PreparedStatement (line 13 in Listing 5).

I've defined SelectStatement to be abstract because it can't define the method ParseResultSet, which is intended to convert the results of a query into Java objects and differs depending upon the data for which you are querying. The parseResultSet method will be dependent on the SQL statement passed to the SelectStatement constructor. To construct a SelectStatement, you need to extend SelectStatement and define the parseResultSet method.

Listing 6 shows a method that queries the database for dealer. This method exists on a class called DealerDAO whose sole purpose is to read and write dealer information into the database. In this example, a dealer contains an id (BigDecimal), a name and a number (both Strings). A JavaBean called DealerVO represents the (value object pattern).

Additional Patterns: I've Introduced Two New Patterns

- **Data access object (DAO) pattern:** A class responsible for reading and writing certain information to the database. It hides the SQL specific code behind a clean interface. It contains methods such as create, remove, update, findByPrimaryKey and additional find Methods. It presents an object-oriented view of the data in the database. In this example, DealerDAO is responsible for writing dealers into the database.
- **Value object (VO) pattern:** Represents data in the database as a JavaBean. Most often represents a row in a table or view (though it could represent a more complicated mapping). Makes the passing of this data between the application tiers easier. In this example, a dealer is represented by a DealerVO value object. DealerVO has three properties: id, name and number.

I've chosen to extend SelectStatement with an anonymous class whose declaration begins on line 7 of Listing 6 because this statement will only be used by the DAO's create method. Should I need it elsewhere, I can redefine it as an inner class and referred to it in both places.

I call the constructor of the SelectStatement and pass it the SQL statement to execute as well as the primary key (wrapped in a List). I further define the method parseResultSet (line 12) to convert each row of the results into a DealerVO by calling the getDealerVOFromResultSet method.

The getDealerVOFromResultSet method is defined on the class DealerDAO not on the Statement class (non-static inner or anonymous classes have access to the member functions of the class in which they are

defined). This is a good architectural decision. The code for converting the ResultSet to a Java object will live in one method and be accessible to any other SelectStatement contained in the DealerDAO. Since the DealerDAO is the only object responsible for reading dealers from the database, no other class should need this method.

Line 25 executes the newly constructed statement object that returns the List returned from parseResultSet on line 22. It pulls out the DealerVO found, if any, and returns it to the calling function.

Updating the Database

For inserting, updating and deleting data in the database, I've created a sub-class of BaseStatement called ModifyStatement, and I've defined the executeStatement() method (line 9 of Listing 7) to call the execute() method off the PreparedStatement passed to it. executeStatement() is defined as returning a ResultSet, and because modifications to the database don't create a ResultSet, it returns null. This is OK because I've also defined the method parseResultSet (which is the only method that uses the ResultSet) to be empty. ModifyStatement is not abstract. It doesn't need to parse a ResultSet (which would make it dependent on the SQL statement passed to the constructor) so all of its methods can be defined.

To Execute a modify statement, you would instantiate a ModifyStatement and pass to it the SQL statement to execute as well as the arguments to insert into the SQL statement.

Listing 8 is the create method defined on the DealerDAO. It takes as its argument a DealerVO (containing the id, name and number of the dealer to be inserted). (Code fragments 8–11 can be downloaded from www.sys-con.com/java/sourcecc.cfm.)

Line 4 declares a String object containing the SQL statement text. Notice the order of the columns. Below that (on line 9), I create a List to store the

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



arguments for the Statement. Notice the order of the arguments. It matches the order of the columns in the SQL text. Specifying both the SQL statement and the arguments in the same place makes aligning their order easier. The ModifyStatement inherits the processStatement method defined on BaseStatement, which will place the arguments from the List into the SQL statement in order.

Lines 14 and 15 construct the ModifyStatement and line 17 executes it.

Managing Transactions

The Statement class can manage transactions itself or let them be managed by the calling method. In the examples we've covered so far, the statement object manages transactions. In the constructor for BaseStatement, I set the boolean member variable commitTransaction to true. Then, the execute() method commits the transaction if it's true. Given this approach, a call to the method create on the DealerDAO, which in turn executes a ModifyStatement to create a dealer, will commit the changes before returning to the user.

However, the calling method may want to execute many Statements and commit them only if all succeed. In this case, the calling method has to manage the transaction.

In Listing 9, I've added a constructor to the BaseStatement class that takes a Connection object as well as the SQL text and argument list. On line 9, I set the commitTransaction flag to false, which will be checked by the Statement's execute() method to determine if it should commit the statement. This lets the calling method handle transactions by passing in the connection object for the Statement to use. The calling method could execute several Statements and commit them only after all of them have succeeded.

I've overloaded the DealerDAO's create method further to take a Connection as well as a DealerVO. It then passes this connection to the ModifyStatement (using the constructor shown in Listing 9). Dealers created this way aren't committed to the database until the calling method calls commit() on its Connection. The calling method can write several dealers to the database (by calling the DealerDAO's create method several times) and commit them all at once. The original create method now calls this new create method passing a null as the connection. This causes the new create method to instantiate a ModifyStatement that will manage the transaction itself (committing all changes).

Thread Safety

I've ensured the thread safety of the Statement objects by: 1.)

making the execute method synchronized, and 2.) making all variables (except the Connection), local to the execute method. No two Statements share variables so multiple Statements can execute without interfering with each other. The one exception is the Connection object. It's stored as a member variable because it can be passed in at construction time. Two statements sharing the same connection could execute simultaneously and affect each other (by committing or closing the connection). In this case, the calling method must ensure thread safety.

Exception Handling

Another benefit of the Template method approach to JDBC is that exception handling is located in one place. In code fragment 1, each of the steps to executing a Statement is wrapped in a try-catch block. I omitted the catch logic for the sake of brevity. Listing 11 contains all the code to the execute method. On line 19, the method catches any exception thrown during the course of executing a Statement and rethrows it as a DAOException (a simple wrapper around the original exception). Calling methods can choose to catch this or allow it to wind further up the stack.

The execute method cleans up all the database resources in a finally block. The ResultSet and PreparedStatement are closed, releasing any resources that they may be holding. If the Statement is managing transactions, the Connection is closed and returned to the pool.

Conclusion

Using the approach described above, database access is as simple as instantiating a class. The JDBC specific code, as well as the exception handling and synchronization, are primarily contained in one class. The developer only has to provide the SQL statement and work with the result set (on database reads). Flexibility is maintained because each step in executing a statement can be redefined through a sub-class, and furthermore, the DAO pattern can be used to wrap the statement thereby hiding the schema dependent code. ☺

References

- *JDBC*: <http://java.sun.com/products/jdbc/>
- *Template Method Pattern*: Design Patterns by Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides
- *DAO pattern*: <http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html>
- *Value Object Pattern*: <http://java.sun.com/j2ee/patterns/ValueObject.html>

Listing 1 The execute method of the class BaseStatement

```
1 public synchronized List execute()
2 throws DAOException {
3
4 List results = new ArrayList();
5 PreparedStatement preparedStatement
6 = null;
7 ResultSet resultSet = null;
8
9 try {
10
11 getConnection();
12 preparedStatement =
13 getPreparedStatement();
14
15 processStatement(preparedStatement);
16 resultSet =
17 executeStatement(preparedStatement);
18 results =
```

```
19 parseResultSet(resultSet);
20
21 if (commitTransaction)
22 connection.commit();
23
24 } catch (Exception e) {
25 // Exception handling omitted for the sake of brevity.
26 }
27 return results;
28 }
```

Listing 2 Getting the connection

```
1 protected Connection getConnection()
2 throws NamingException, SQLException {
3
4 if (connection == null) {
5 Context jndiContext = new
```

```

6 InitialContext();
7
8     DataSource ds =
9     (DataSource)
10    jndiContext.lookup
11
12    (JNDINames.DATASOURCEJNDINAME);
13
14    connection = ds.getConnection();
15    connection.setAutoCommit(false);
16 }

return connection;
}

```

Listing 3 Getting and processing the statement.

```

1 protected PreparedStatement
2 getPreparedStatement()
3 throws Exception {
4
5 return
6 connection.prepareStatement(preparedStateM
7 entText);
8 }

9
10 protected void processStatement(PreparedStatement
11 preparedStatement)
12 throws Exception {
13 for (int i = 0; i < arguments.size(); i++) {
14     preparedStatement.setObject(i + 1, arguments.get(i));
15 }
16 }

```

Listing 4

```

1 public BaseStatement(String
2 preparedStatementText, List arguments) {
3
4 this.preparedStatementText =
5 preparedStatementText;
6 this.arguments = arguments;
7 this.commitTransaction = true;
8 }

```

Listing 5

```

1 public abstract class SelectStatement
2 extends BaseStatement {
3
4 public SelectStatement(
5     String selectStatementText, List
6 args) {
7
8 super(selectStatementText, args);
9 }
10
11
12 protected ResultSet
13 executeStatement(
14     PreparedStatement
15 preparedStatement) throws Exception {
16
17 return
18 preparedStatement.executeQuery();
19 }
20 }

```

Listing 6

```

1 public DealerVO
2 findByPrimaryKey(BigDecimal primaryKey)
3 throws DAOException {
4
5 List parameters = new ArrayList();
6 parameters.add(primaryKey);
7
8 SelectStatement selectStatement =
9 new SelectStatement(
10 "Select * from MFS_DEALERS where
11 DEALER_ID = ?",
12 parameters) {
13
14 public List parseResultSet(
15     ResultSet resultSet) throws Exception {
16
17
18     List results = new A
19 rrayList();
20
21 if (resultSet.next()) {
22     results.add(
23
24     getDealerVOFromResultSet(resultSet));

```

```

25 }
26
27 return results;
28 }
29 };
30 List results =
31 selectStatement.execute();

if (results != null && results.size() > 0)
return (DealerVO) results.get(0);
else
return null;
}

```

Listing 7

```

1 public class ModifyStatement extends
2 BaseStatement {
3
4 public ModifyStatement(
5     String preparedStatementText,
6     List arguments) {
7 super(preparedStatementText,
8 arguments);
9 }
10
11 protected ResultSet
12 executeStatement(
13     PreparedStatement
14 preparedStatement)
15 throws Exception {
16
17     preparedStatement.execute();
18
19 return null;
20 }
21
22     protected List
23 parseResultSet(ResultSet resultSet)
24 throws Exception {
25 return null;
26 }
27 }

```



The Java™ Presentation framework
for J2EE™ Web applications

Based on:

- Java™
- Servlets™
- Java Serverpages™
- and Struts



Get your free trial version
– www.common-controls.com
See the common controls in action
– go for the Online Demo!

Contains the most common control elements
which are required for the development of J2EE™
applications with rich HTML frontends like:



www.common-controls.com

Which logging library is better for you?

by Joe McNamara

Are your Java programs littered with a multitude of randomly placed System.out.println statements and stack traces? When you add debugging messages to a class in a project, are the outputs of your messages interleaved among dozens of messages from other developers, making your messages difficult to read? Do you use a simple, hand-rolled logging API, and fear that it may not provide the flexibility and power that you need once your applications are in production? If you answered yes to any of the above questions, it's time for you to pick an industrial-strength logging API and start using it.

This article will help you choose a logging API by evaluating two of the most widely used Java logging libraries: the Apache Group's Log4j and the java.util.logging package (referred to as "JUL"). This article examines how each library approaches logging, evaluates their differences and similarities, and offers a few simple guidelines that will help you decide which library to choose.



Joe McNamara is a software developer and logging guru at Quantum Leap Innovations, an innovator of intelligent software. At Quantum Leap Innovations, he works on a revolutionary multiagent system technology for the seamless and dynamic integration of wide numbers of applications, systems, and human users.

jem@quantumleap.us

Introduction to Log4j

Log4j is an open source logging library developed as a sub-project of the Apache Software Foundation's Logging Services Project. Based on a logging library developed at IBM in the late 1990s, its first versions appeared in 1999. Log4j is widely used in the open source community, including by some big name projects such as JBoss and Hibernate.

Log4j's architecture is built around three main concepts: loggers, appenders, and layouts. These concepts allow developers to log messages according to their type and priority, and to control where messages end up and how they look when they get there. Loggers are objects that your applications first call on to initiate the logging of a message. When given a message to log, loggers generate LoggingEvent objects to wrap the given message. The loggers then

hand off the LoggingEvents to their associated appenders. Appenders send the information contained by the LoggingEvents to specified output destinations – for example, a ConsoleAppender will write the information to System.out, or a FileAppender will append it to a log file. Before sending LoggingEvent information to its final output target, some appenders use layouts to create a text representation of the information in a desired format. For example, Log4j includes an XMLLayout class that can be used to format LoggingEvents as strings of XML.

In Log4j, LoggingEvents are assigned a level that indicates their priority. The default levels in Log4j are (ordered from highest to lowest): OFF, FATAL, ERROR, WARN, INFO, DEBUG, and ALL. Loggers and appenders are also assigned a level, and will only execute logging requests that have a level that is equal to or greater than their own. For example, if an appender whose level is ERROR is asked to write out a LoggingEvent that has a level of WARN, the appender will not write out the given LogEvent.

All loggers in Log4j have a name. Log4j organizes logger instances in a tree structure according to their names the same way packages are organized in the Java language. As Log4j's documentation succinctly states: "A logger is said to be an ancestor of another logger if its name followed by a dot is a prefix of the descendant logger name. A logger is said to be a parent of a child logger if there are no ancestors between itself and the descendant logger." For example, a logger named "org.nrdc" is said to be the child of the "org" logger. The "org.nrdc.logging" logger is the child of the "org.nrdc" logger and the grandchild of the "org" logger. If a logger is not explicitly assigned a level, it uses the level of its closest ancestor that has been assigned a level. Loggers inherit appenders from their ancestors, although they can also be configured to use only appenders that are directly assigned to them.

When a logger is asked to log a message, it first checks that the level of the request is greater than or equal to its effective level. If so, it creates a LoggingEvent from the given message and passes the LoggingEvent to its appenders, which format it and send it to its output destinations.

Introduction to JUL

The java.util.logging package, which Sun introduced in 2002 with Java SDK version 1.4, came about as a result of

JSR 47, Logging API Specification. JUL is extremely similar to Log4j – it more or less uses exactly the same concepts, but renames some of them. For example, appenders are “handlers,” layouts are “formatters,” and LoggingEvents are “LogRecords.” Figure 1 summarizes Log4j and JUL names and concepts. JUL uses levels the same way that Log4J uses levels, although JUL has nine default levels instead of seven. JUL organizes loggers in a hierarchy the same way Log4j organizes its loggers, and JUL loggers inherit properties from their parent loggers in more or less the same way that Log4j loggers inherit properties from their parents. Concepts pretty much map one-to-one from Log4j to JUL; though the two libraries are different in subtle ways, any developer familiar with Log4j needs only to adjust his or her vocabulary to generally understand JUL.

Functionality Differences

While Log4j and JUL are almost conceptually identical, they do differ in terms of functionality. Their difference can be summarized as, “Whatever JUL can do, Log4j can also do – and more.” They differ most in the areas of useful appender/handler implementations, useful formatter/layout implementations, and configuration flexibility.

JUL contains four concrete handler implementations, while Log4j includes over a dozen appender implementations. JUL’s handlers are adequate for basic logging – they allow you to write to a buffer, to a console, to a socket, and to a file. Log4j’s appenders, on the other hand, probably cover every logging output destination that you could think of. They can write to an NT event log or a Unix syslog, or even send e-mail. Figure 2 provides a summary of JUL’s handlers and Log4j’s appenders.

JUL contains two formatter classes: the XMLFormatter and SimpleFormatter. Log4j includes the corresponding layouts: the XMLLayout and SimpleLayout. Log4j also offers the TTCCLayout, which formats LoggingEvents into content-rich strings, and the HTMLLayout, which formats LoggingEvents as an HTML table.

While the TTCCLayout and HTMLLayout are useful, Log4j really pulls ahead of JUL in the formatter/handler arena because of the PatternLayout. PatternLayout instances can be configured with an enormous amount of flexibility via string conversion patterns, similar to the conversion patterns used by the printf function in C. In PatternLayout conversion patterns, special conversion characters are used to specify the information included in layout’s formatted output. For example, “%t” is used to specify the name of the thread that started the logging of the message; “%C” is used to specify the name of the class of the object that started the logging of the message; and “%m” specifies the message. “%t: %m” would result in output such as “main thread: This is my message.” “%C - %t: %m” would result in output such as “org.nrdc.MyClass - main thread: This is my message.” The PatternLayout is extremely useful, and JUL’s two formatter classes don’t come anywhere near to matching its versatility. It’s not uncommon for JUL users to write their own custom formatter class, whereas most Log4j users generally need to just learn how to use PatternLayout conversion patterns.

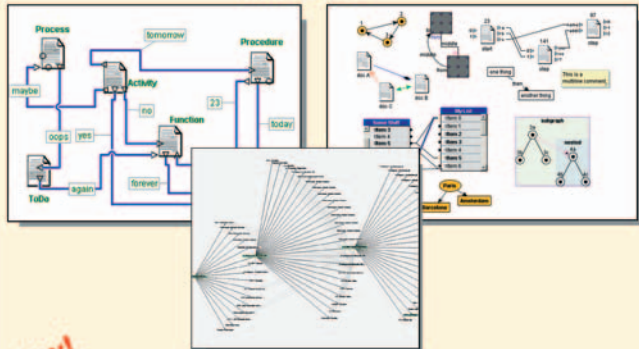
While both Log4j and JUL can be configured with configuration files, Log4j allows for a broader range of configuration

Log4j Name	JUL Name	Purpose
Logger	Logger	Entrance point from outside applications into logging framework; handles requests to log messages
Appender	Handler	Receives LoggingEvents/LogRecords from Loggers, formats them with a Layout/Formatter, and send them to their output destination
Layout	Formatter	Converts the information contained in a LoggingEvent/LogRecord into a String of a desired format
LoggingEvent	LogRecord	An Object that encapsulates a message to be logged; contains the message as well as other information such as the message’s priority (Level) and the Class where the message is being logged from.
Level	Level	Indicates the priority of a LoggingEvent/LogRecord; also, Loggers and Appenders/Handlers have a Level; stipulating the threshold for LoggingEvents/LogRecords that they will log
FIGURE 1		LOG4J AND JUL IMPORTANT CONCEPTS AT-A-GLANCE

possibilities through configuration files than JUL does. JUL can be configured with .properties files, but until J2SE 5.0 the configuration of handlers was only on a per-class rather than a per-instance basis. This means that if you are going to be using a pre-Tiger SDK, you’ll miss out on useful configuration options, such as the ability to set up different FileHandler instances to send their output to different files.

It’s important to note that pre-Tiger JUL can easily be configured to write to multiple output files in code, just not through its default configuration mechanism. Log4j can be configured with .properties and/or XML files, and appenders can be configured on a per-instance basis. Also, Log4j

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



allows developers to associate layout instances with appender instances, and configure layouts on a per-instance basis. This includes PatternLayout instances – you can set the conversion pattern each uses in the configuration file. During development, it usually isn't a problem to recompile an application to adjust its logging configuration; after deployment, however, you may want to be able to tweak or even completely reconfigure your application's logging without recompiling. In that case, Log4j offers more flexibility, especially pre-Tiger.

Log4j provides a lot of functionality that JUL lacks, although JUL is catching up. JUL could definitely be extended to do what Log4j does – you could write more handlers, reimplement the PatternLayout for JUL, and upgrade JUL's configuration mechanism, all without extreme difficulty. But why do that when Log4j has had those features for years?

Which Library Do You Choose?

Important decisions such as these typically make project leaders lose sleep and go prematurely gray. Luckily, this decision can be made very easily by examining the answers to three simple questions.

Question One

Do you anticipate a need for any of the clever handlers that Log4j has that JUL does not have, such as the SMTPHandler, NTEventLogHandler, or any of the very convenient FileHandlers?

Question Two

Do you see yourself wanting to frequently switch the format of your logging output? Will you need an easy, flexible way to do so? In other words, do you need Log4j's PatternLayout?

Question Three

Do you anticipate a definite need for the ability to change complex logging configurations in your applications, after they are compiled and deployed in a production environment? Does your configuration sound something like, "Severe messages from this class get sent via e-mail to the support guy; severe messages from a subset of classes get logged to a syslog daemon on our server; warning messages from another subset of classes get logged to a file on network drive A; and then all messages from everywhere get logged to a file on network drive B"? And do you see yourself tweaking it every couple of days?

If you can answer yes to any of the above questions, go with Log4j. If you answer a definite no to all of them, JUL will be more than adequate and it's conveniently already included in the SDK.

Conclusion

Log4j and JUL are very similar APIs. They differ conceptually only in small details, and in the end do more or less the same thing, except Log4j has more features that you may or may not need.

Keep in mind as you migrate to your chosen logging library that logging may affect the performance of your application. Make its impact as light as possible by reusing references to loggers; keep a static or instance pointer to loggers that you use, rather than calling `Logger.getLogger("loggerName")` every time you need a logger. Use common sense in your placement of log statements – do not place them in tight, heavily iterated loops.

This article is not an in-depth tutorial on how to use Log4j or JUL, and, in fact, has glossed over a number of useful features in both libraries, such as MBean support (in J2SE 5.0, you'll be able to set JUL logging levels remotely using JMX), and ResourceBundle support. There are also a number of advanced features that only Log4j has, such as filter chaining and ObjectRenderers. The Internet is full of great tutorials on how to use both libraries, including a number of articles in *JDJ's* archives; be sure to check them out before you begin coding. ☺

Resources

- *Log4j's home page*: <http://logging.apache.org/log4j>
- *JUL's home page*: <http://java.sun.com/j2se/1.4.2/docs/guide/util/logging>
- Aggarwal, V. "Third Party Logging API." *Java Developer's Journal*, Vol. 5, issue 11: <http://sys-con.com/story/?storyid=36144>
- Banes, J. "Building the Ultimate Logging Solution." *Java Developer's Journal*, Vol. 9, issue 5: <http://sys-con.com/story/?storyid=44698>
- Writing a sweet Log4j Appender that sends instant messages: www-106.ibm.com/developerworks/java/library/j-instlog/
- For those who don't like JUL or Log4j, try the Logging Toolkit for Java from IBM: www.alphaworks.ibm.com/tech/loggingtoolkitj

JUL Handler Classes

Class Name	Function
ConsoleHandler	Writes to System.out or System.err
MemoryHandler	Writes to a Buffer, can be used with other Handlers
SocketHandler	Writes to a socket on a specified port
File Handler	Writes to a specified file

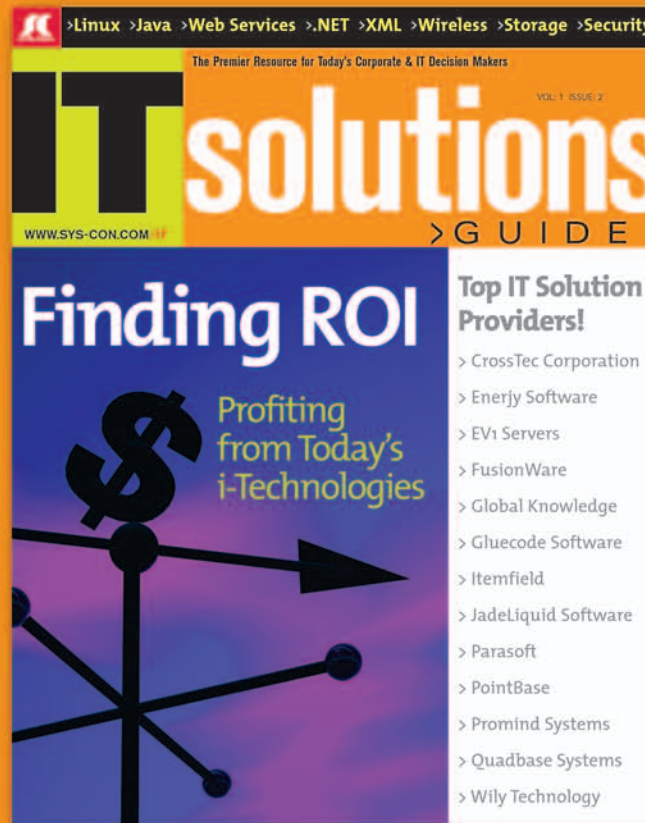
Some Useful Log4j Appender Classes

Class Name	Function
ConsoleAppender	Writes to System.out or System.err
FileAppender	Writes to a specified file
DailyRollingFileAppender	Writes to a file; automatically rolls over to a new log file daily
ExternallyRolledFileAppender	Listens on a specified port for a "rollover" message, rolls over at each receipt of message.
JDBCAppender	Logs to a JDBC connection
JMS Appender	Publishes LoggingEvents to a specified JMS topic
NTEventLogAppender	Writes to an NT event log
SMTPAppender	Sends email containing log messages
SocketAppender	Writes to a socket on a specified port
SocketHubAppender	Sets up server on specified port that allows clients to connect; sends log messages to all connected clients
SyslogAppender	Sends log messages to a remote Unix syslog daemon
TelnetAppender	Sets up server on specified port; clients using Telnet can connect to receive log messages

FIGURE 2 LOG4J INCLUDES SOME VERY CLEVER AND USEFUL APPENDER IMPLEMENTATIONS

Reach Over 100,000

Enterprise Development Managers & Decision Makers with...



Offering leading software, services, and hardware vendors an opportunity to speak to over 100,000 purchasing decision makers about their products, the enterprise IT marketplace, and emerging trends critical to developers, programmers, and IT management

Don't Miss Your Opportunity to Be a Part of the Next Issue!

Get Listed as a Top 20* Solutions Provider

For Advertising Details Call 201 802-3021 Today!

*ONLY 20 ADVERTISERS WILL BE DISPLAYED. FIRST COME FIRST SERVE.



The World's Leading i-Technology Publisher



Joe Winchester
Desktop Java Editor



Go Fast It Runs Too Slow

Go fast, it runs too slow, you've got to make the number show. Diddle de bop, da la de doop, sitting around and feeling groovy.

Speed Is as Speed Does

Many moons ago I was working on a project that had to be sped up and we had the benefit of a very experienced consultant to help us out. Fresh from his business-class flight and clutching his pay-as-you-go expense account lunch, our management team eagerly led him over to where our assembled developers waited in awe (and with a certain amount of natural coder-sapiens resistance to the hired gun who'd come to town to sheriff us).

Like a surgeon approaching a sick patient we expected a briefcase to be opened revealing dozens of tools for analyzing memory leakage, profiling garbage collection across multiple threads, searching out deadlocks and everything else we assumed was slowing down the program. Instead all he produced was a simple red stopwatch.

All that matters to the user is how long a given scenario takes and whether it's perceived as slow or not. The first thing to do was to identify which tasks were too slow, figure out what response time would be acceptable, and then keep working back against that benchmark.

Apples and Apples

The idea is to focus on real user scenarios and their actual end-to-end response times. Instead of our zippy high-end boxes, dust off older, slower machines and dedicate them to doing the benchmarks. Us spoiled developers often run the latest wizzo kit and don't appreciate the realities of the boxes our code actually runs (or walks on) in the field.

Look Under the Hood

Having identified the slow scenarios,

the next step is to identify where the time is going. Probably no single tool can yield all that information, and techniques can vary from simple tracelogs of the current clock time at various stages in the program through to powerful local and total time method breakdowns. Garbage collection, file I/O, page faults, all need to be observed and understood.

Fix What's Broke, Forget What's Not

Looking at the analysis from the running application can yield a bunch of fix candidates. But rather than jumping in and starting to redesign everything, it's good to prove first that there will be a real benefit before doing any work. Like writing a unit test before programming, it may be better to create dummy fixes that simulate the corrected code without being functionally complete. The exercise might involve changing a method to hard coding a result instead of an expensive piece of computation, or putting data in a stale cache and running performance tests to see if the fancy auto-rebuilding dancing cache is really worth doing. Before creating a solution, you have to know there's a real problem as opposed to a perceived one you just feel like polishing. Patch fixes can be created for dummy fixes, applied and run against a full build on the benchmark box to see what affect they have and, by extrapolation, the benefit fully functional fixes will bring to the bottom line.

Optimize Code Paths

Sometimes single method calls result in an explosion of code where a leaf method is being executed thousands of times. With the execution time of the lowest-level method being gauged in fractions of milliseconds, this becomes significant. The solution is to rewrite the code path to avoid such a deluge. The change itself might not be particularly difficult but without a good tool to get the problem on the radar, perhaps as

a result of an n^2 or even n^3 to the n algorithm, it might never even be considered a possible problem.

Minimize IO

One way to produce big performance improvements is to reduce the amount of file reads and writes and socket access. Some data needs to be re-read frequently because it's volatile, but objects such as icons or definition files are good candidates to access once and cache in library registries. Socket I/O is another place to look as well. We found that the problem with one application was in the latency of the conversation, not in the amount of data being sent across the wire. It was overcome by batching data packets together into larger-grained messages.

Cache Model Data

Using a cache to speed something up can be either a silver bullet or fool's gold. The latter is like someone who claims they get great mileage from their car by not driving it. If the program is fast enough to begin with, caches aren't needed, so the first port of call should be to see if there are other unexplored avenues to make things quicker. If a cache is required then it comes with the health warning that you not only have to build it, you need to know when the data you're caching might become stale.

Leave Your Assumptions at the Door

One of the most startling things about tuning a program is that sometimes a lot of fancy code that was initially designed to help performance isn't that useful. Ironically the clever algorithms might actually cause harm. While they may add nothing significant to the bottom line, they can make the code harder to read and understand and affect its maintainability. This is part of the philosophy behind the XP mantra "Make it work, make it right, make it fast." Until you know what's slow, don't try to make it faster. ☺

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

joewinchester@sys-con.com

Advertiser	URL	Phone	Page
Altova	www.altova.com	978-816-1600	4
Business Objects	www.businessobjects.com/dev/p26	888-333-6007	26-27
ceTe Software	www.dynamicpdf.com	800-631-5006	33
Common Controls	www.common-controls.com	+49 (0) 6151/13 6 31-0	45
DataDirect	www.datadirect.com/jdj	800-876-3101	Cover IV
EV1 Servers	www.ev1servers.net	800-504-SURF	23
Fitech Laboratories	www.fitechlabs.com/grid	646-495-5076	37
Google	www.google.com/jdj	650-623-4000	43
Information Storage & Security Journal	www.issjournal.com	888-303-5282	51
IT Solutions Guide	www.sys-con.com/it	888-303-5282	49
Java Developer's Journal	www.sys-con.com/jdj	888-303-5282	55
Jinfony Software	www.jinfony.com/jp3.htm	301-838-5560	15
M7	www.m7.com/d7.do	866-770-9770	21
Microsoft	www.msdn.microsoft.com/visual		9
Nexaweb	www.nexaweb.com		31
Northwoods Software Corp.	www.nwoods.com/go	800-434-9820	47
Parasoft Corporation	www.parasoft.com/jtest_IDJ	888-305-0041	7
Phoneomena	www.phoneomena.com	352-373-3966	35
ReportingEngines	www.reportingengines.com	888-884-8665	17
SAP	www.sdn.sap.com		25
Sleepycat Software	www.sleepycat.com/bdbje	510-597-2128	39
Software FX	www.softwarefx.com	800-392-4278	Cover III
Sun Microsystems	www.developers.com/prodtech/javatools/jenterprise/downloads		19
WebRender	www.webrender.com	+61 3 6226 6274	41
ZeroG Software	www.zerog.com	415-512-7771	Cover II

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

Making PDFs Portable

Integrating PDF and Java technology

by Ben Litchfield

Since Adobe released the first public PDF Reference in 1993, a number of PDF utilities and libraries, supporting all kinds of languages and platforms, have been made available to users and developers alike. However, support for Adobe's technology has lagged in Java application development. And this is curious because PDF documents tend to be a popular way of storing and interchanging information when dealing with enterprise information systems – an application domain that Java technology is particularly well suited to. Yet it seems that, until recently, mature, capable PDF support wasn't readily available to Java applications developers.

PDFBox (an Open Source project released under the BSD license) is a pure Java library that lets developers read and create PDF documents. It has features such as:

- Extracting text, including Unicode characters
- Easy integration with text search engines like Jakarta Lucene
- Encryption/Decryption of PDF documents
- Importing/Exporting of form data in FDF and XPDF formats
- Appending to existing PDF documents
- Splitting a single PDF into multiple documents
- Overlaying one PDF document on top of another

PDFBox API

PDFBox has been designed to represent PDF documents using familiar object-oriented paradigms. The data contained in a PDF document is a collection of basic object types: arrays, booleans, dictionaries, numbers, strings and binary streams. PDFBox captures these basic object types in the *org.pdfbox.cos* package (the COS Model). While it's possible to create any desired interactions

with a PDF document using only these objects, it requires an intimate knowledge of the internals of PDF documents and the techniques used to represent higher-level concepts. For example, objects such as pages and fonts are represented as dictionaries with specialized attributes; deciphering all these various attributes and their types requires tedious consultation of the PDF Reference.

For this reason, the *org.pdfbox.pdmodel* package (the PD Model) sits on top the COS Model and provides a high-level API that accesses PDF document objects in a more familiar manner (see Figure 1). Objects such as *PDPage* and *PDFont* can be found in this package, which encapsulates their lower-level COS model counterparts.

A word of caution to developers: the PD Model offers many nice features but is still a work in progress. In some instances, use of the COS Model may be required to access a particular piece of PDF functionality. Consequently, all PD Model objects can retrieve the corresponding

COS Model object that they represent, so it's always possible to start with the PD Model and drop down to the COS Model when the required piece of functionality is found to be missing.

Now that the general capabilities of PDFBox have been discussed a few examples of its use are appropriate. We will start by reading an existing PDF document:

```
PDDocument document =
    PDDocument.load( "./test.pdf" );
```

This operation will cause the PDF file to be parsed and an in-memory representation of the document will be created. To facilitate the efficient handling of large documents, PDFBox only stores the document structure in memory; objects such as images, embedded fonts and page content are cached in a temporary file.

Note: When finished using a *PDDocument* object, care should be taken to invoke the *close()* method on the document object to release resources used during its creation.



Ben Litchfield is a business systems consultant in the development and integration practice at LPA Systems. He has been the lead developer of PDFBox for the past two years. Ben holds a B.S. in software engineering from the Rochester Institute of Technology and has been providing enterprise application solutions for the past five years.

ben.litchfield@lpasystems.com

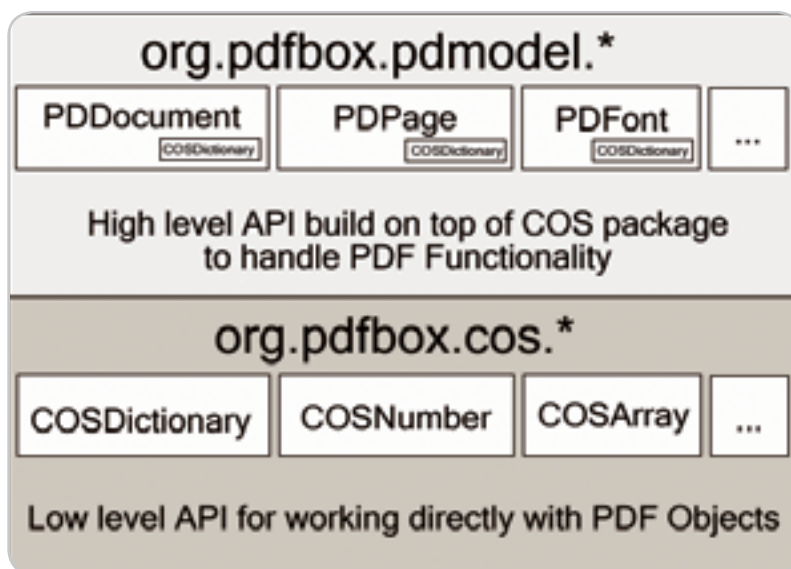


Figure 1 PD Model and COS Model diagram

Text Extraction and Lucene Integration

In an information retrieval age when applications are expected to have searching and indexing capabilities regardless of the medium, the ability to organize and catalog information into a searchable format is critical. This is simple for text and HTML documents, but PDF documents have more structure and meta-information that makes it difficult to extract the underlying text. The PDF language is similar to Postscript in that objects are drawn as vectors on the page at certain positions. For example:

```
/Helv 12 Tf
0 13.0847 Td
(Hello World) Tj
```

This set of instructions changes the font to Helvetica size 12, moves the caret to the next line and renders the string “Hello World.” These command streams are usually compressed and the order in which the glyphs are displayed on the screen is not necessarily the order in which the characters appear in the file, so it isn’t always possible to simply extract text strings directly from the raw PDF document. However, PDFBox has a sophisticated text-extraction algorithm that deals with this and other complexities, letting a developer get the text of the document as if reading off its rendered form.

Lucene, which is part of the Apache Jakarta project, is a popular Open Source search engine library. Lucene lets developers create an index and do complex searches on a large volume of textual content based on that index. Since Lucene has adopted text as the common denominator for content, it’s the developer’s responsibility to convert the data contained in other desired file formats to text to use Lucene. For example, file formats such as Microsoft Word and StarOffice documents have to be converted to text before they can be added to a Lucene index.

PDF files are no exception, but PDFBox makes it easy to include a PDF document in a Lucene index by supplying a special object that does the integration. A basic PDF document can be converted to a Lucene document with a single statement:

```
Document doc =
    LucenePDFDocument.getDocument( file );
```

This operation parses the PDF document, extracts the text and creates a Lucene document object that can then be added to the index. As mentioned above, PDF documents also contain metadata such as author information and keywords that are important to track when indexing PDF documents. Table 1 shows the fields that PDFBox will populate while creating the Lucene document.

This integration makes it easy for developers to support simple searching and indexing of PDF documents with Lucene. Of course, some applications require more sophisticated text-extraction methods. In that case, the *PDFTextStripper* class can be used directly, or extended to handle these complex requirements.

By extending this class and overriding the *showCharacter()* method, many aspects of text extraction can be con-

Lucene Field Name	Description
path	Filesystem path if loaded from a file
url	URL to PDF
contents	Entire text content of PDF document
summary	First 500 characters of document
modified	The date/time of the PDF according to the file or URL
uid	A unique identifier for Lucene
CreationDate	From PDF metadata
Creator	From PDF metadata
Keywords	From PDF metadata
ModificationDate	From PDF metadata
Producer	From PDF metadata
Subject	From PDF metadata
Trapped	From PDF metadata

Table 1 Lucene Fields populated by PDFBox

trolled. For instance, an implementation of this method can use the x, y positioning information to limit the inclusion of certain blocks of text in the extraction. One use might exclude all of the text above a certain y-coordinate value effectively excluding an unwanted document header.

Another example: Oftentimes a group of PDF documents may have been created from forms and the source data are no longer available. In other words, the documents all have some interesting text at similar locations on the page, but the form data used to fill the document out are no longer available. For example, a collection of cover letters that have the name and address at the same location in the document. In this case, an extension of the *PDFTextStripper* class can be used as a sort of screen-scraping device to extract the desired fields.

Encryption/Decryption

A popular PDF feature allows for encrypting document contents and setting access controls limiting who can view the unencrypted document. Specifically, a PDF document is encrypted with a master password and optionally a user password. If a user password has been provided, then a PDF reader such as Acrobat will prompt for a password before letting the document be viewed. The master password is required to change document permissions.

The PDF specification lets creators of PDF documents restrict certain operations when viewing the PDF in Acrobat. Some of the available document restrictions are:

- Printing
- Changing content
- Extracting text

A full explanation of PDF document security lies outside the bounds of this article and interested developers should reference the relevant sections of the PDF specification and evaluate its capabilities. The security model used in PDF documents is pluggable and lets different security handlers be employed when encrypting documents. As of this writing, PDFBox supports the “Standard” security handler, which is what most PDF documents use.

To encrypt a document, it must first be assigned a security handler and then encrypted with a master password and user password. For example, the following code encrypts a document so a user can open it in Acrobat without entering a password (i.e., no user password), but can't print the document using the access control mechanism.

```
//load the document
PDDocument pdf =
    PDDocument.load( "test.pdf" );
//create the encryption options
PDStandardEncryption encryptionOptions =
    new PDStandardEncryption();
encryptionOptions.setCanPrint( false );
pdf.setEncryptionDictionary(
    encryptionOptions );
//encrypt the document
pdf.encrypt( "master", null );
//save the encrypted document
//to the file system
pdf.save( "test-output.pdf");
```



Figure 2 PDFViewer

Utility	Description
org.pdfbox.Decrypt	Decrypt a PDF document, requires the master password.
org.pdfbox.Encrypt	Encrypt a PDF document.
org.pdfbox.ExportFDF	Export form data in FDF format.
org.pdfbox.ExportXFDF	Export form data in XFDF format.
org.pdfbox.ExtractText	Extract text from a PDF document.
org.pdfbox.ImportFDF	Import form data in FDF format.
org.pdfbox.ImportXFDF	Import form data in XFDF format.
org.pdfbox.Overlay	Overlay one document on top of another, useful for a template PDF that contains a header or footer.
org.pdfbox.PDFSplit	Split a multi-page PDF into a series of single page PDF documents. Uses Splitter object, a class that can be extended to define where splits occur, the default is every page.
org.pdfbox.PDFViewer	A PDF debugging utility, shows the internal document structure. See Figure 2.

Table 2 PDFBox Utilities

For a more complete example, reference the source code for the encryption utility included in the PDFBox distribution: *org.pdfbox.Encrypt*.

Many applications can generate PDF documents but don't allow control over the document's security options. PDFBox can be used here to intercept and encrypt the PDF before it's sent to the user.

Form Integration

When an application's output is a series of form field values, it is usually desirable to let the user save the form for record keeping. PDF technology is a great choice for this kind of output. A developer can write code to output PDF instructions manually to draw images, tables and text. Or encapsulate the data in XML and use an XSL-FO engine to create a PDF document. However, these approaches can be time-consuming, error-prone and inflexible. A better approach for simple forms might be to create a template and generate a filled-in document for any given set of input data based on the template.

A form many of us may be familiar with is the Employment Eligibility Verification, or I-9 form: <http://uscis.gov/graphics/formsfee/forms/files/i-9.pdf>

Using one of the example applications distributed with PDFBox, the form field names can be listed:

```
java org.pdfbox.examples.fdf.PrintFields
i-9.pdf
```

Another example utility populates a given field with textual data:

```
java org.pdfbox.examples.fdf.SetField i-9.pdf NAME1 Smith
```

Opening the PDF document in Acrobat shows that the "Last Name" field has been filled in. This functionality can be recreated in code:

```
PDDocument pdf =
    PDDocument.load( "i-9.pdf" );
PDDocumentCatalog docCatalog =
    pdf.getDocumentCatalog();
PDAcroForm acroForm =
    docCatalog.getAcroForm();
PDFField field =
    acroForm.getField( "NAME1" );
field.setValue( "Smith" );
pdf.save( "i-9-copy.pdf" );
```

It's also possible to extract the values of a form field that has been previously populated, as below:

```
PDFField field =
    acroForm.getField( "NAME1" );
System.out.println(
    "First Name=" + field.getValue() );
```

Acrobat offers the option of exporting and importing form data in a special file format called "Forms Data Format." These files come in two flavors, FDF and XFDF. An FDF stores the form data in the same format as PDF, while XFDF stores data in XML format. PDFBox handles both FDF and XFDF data with a single object: *FDFDocument*. The following snippet shows how to export FDF data for the I-9 form above:

```
PDDocument pdf =
    PDDocument.load( "i-9.pdf" );
PDDocumentCatalog docCatalog =
    pdf.getDocumentCatalog();
PDAcroForm acroForm =
    docCatalog.getAcroForm();
FDFDocument fdf = acroForm.exportFDF();
fdf.save( "exportedData.fdf" );
```

PDFBox Form Integration Steps

1. Create PDF Form Template using Acrobat or other visual tool
2. Track the name of each desired form field
3. Store the template PDF where the

“Java support for PDF has been spotty, but now the Open Source PDFBox project lets Java developers read and create PDF documents”

- application can access it
4. When the PDF is requested, use PDFBox to parse the template PDF
 5. Populate the required form fields
 6. Stream the PDF back to the user

Utilities

Besides the library APIs mentioned above, PDFBox also has a set of command-line utilities. Table 2 lists the class name of each utility along with a short description.

Remarks

The PDF specification weighs in at 1,172 pages so implementing it is quite an undertaking. As such, PDFBox is distributed with the proviso that it is a work in progress, with new features being added over time. Its main weakness is in creating PDF documents from scratch. However, there are several other Open Source Java projects that can be used to fill the gap. For in-

stance, the Apache FOP project lets programmers generate a PDF from a specialized XML document that describes the PDF document. Also, iText provides a high-level API for creating document elements such as tables and lists.

The next version of PDFBox will add support for the new PDF 1.5 object stream and cross-reference streams. After that will be support for embedding fonts and images. Hopefully through efforts like PDFBox, robust support for PDF technology can be made available for Java applications. ☺

References

- PDFBox: <http://www.pdfbox.org/>
- Apache FOP: <http://xml.apache.org/fop/>
- iText: <http://www.lowagie.com/iText/>
- PDF Reference: <http://partners.adobe.com/asn/tech/pdf/specifications.jsp>
- Jakarta Lucene: <http://jakarta.apache.org/lucene/>



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

Casting Perlin's Movie Magic in Java3D

How did they do that?

by Michael Jacobs

Reach behind your television and yank the cable out of the wall. Do you hear that noise? Not the kids screaming about their movie. Look at the screen. What you see is white noise: random bits of white, black and gray changing constantly. What does this have to do with movie magic or Java3D? What if a spell could conjure roaring fires, fluffy clouds, rippling water, naturally grained wood, smooth marble and even realistic terrains? That spell is available to us thanks to the inventive mind of Dr. Ken Perlin.

Who Was That Math Man?

Ken Perlin is a professor in the department of computer science at New York University. In 1997 he won an Academy Award for Technical Achievement from the Academy of Motion Picture Arts and Sciences for his procedural texturing techniques, which are widely used in feature films and television. He also had a big part in the computer animation in the movie "TRON." The techniques pioneered by Dr. Perlin allow programs to generate a wide range of realistic special effects efficiently. The foundation for these effects is a mathematical function called Perlin noise.



Mike Jacobs is a technical architect working at the Mayo Foundation for Medical Education and Research. He has developed CPU hardware, microcode, application components and applications in the financial and healthcare industries. He has extensive design and implementation experience in object-oriented languages including Smalltalk, C++, and Java.

jacobs.michael@mayo.edu

What Was That Noise?

Procedural texturing is the art of using an algorithm to generate a texture. Procedural techniques are not limited to texturing and can be applied to geometry, motion, color or any other thing you can imagine. Procedural techniques abstract the details of a scene or sequence into an algorithm. Parameters on the algorithm allow a variety of results to be achieved with the same algorithm. An example of procedural geometry was in my last article ("When Mars Is Too Big to Download," JDJ, September 2004) where we used parameters to vary the detail and roughness of the generated terrain. The advantage of using procedural techniques is that the details are generated, saving the cost of explicitly storing and retrieving them.

To make realistic special effects, we need a way to generate natural looking randomness. You might think that random numbers would be sufficient to accomplish this, but you would only be partially right. (See Figure 1.) Random numbers are typically generated without regard to past values. This lack of correlation can lead to abrupt changes between adjacent values and an unnatural special effect. What we need is a repeatable, smooth, non-cyclic random function whose results vary with the parameters we provide. Perlin noise was designed to do just that.

While the implementation details of Perlin noise are beyond the scope of this article, we do need a conceptual model to use it. The noise function accepts a number of double parameters and returns a double value between +1 and -1. One-dimensional noise is the result of generating random numbers at regular intervals and smoothly interpolating noise values in between using a high-order polynomial. This can be represented by a smooth curve as shown in Figure 1. Two-dimensional noise does the interpolation in two dimensions forming a wavy noise surface. The three-dimensional noise can't be depicted graphically, but its foundation is a lattice. The three parameters represent the three dimensions of the lattice from which the noise value is calculated. This

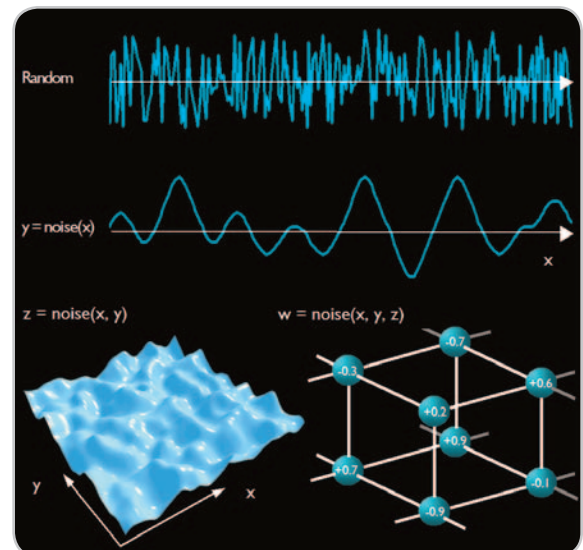


Figure 1 Random numbers change abruptly compared to Perlin noise

lattice consists of 256 by 256 by 256 points representing random numbers between which values are smoothly interpolated to calculate the noise. The noise value at the integer lattice locations is zero, while the values between the locations follows the same high-order polynomial mentioned above. A Java reference implementation called *ImprovedNoise* is available from Dr. Perlin's home page and a modified version is included with the source code for this article.

This probably sounds pretty mysterious, so let's put this magic to work with a few examples.

Casting Our First Spell: Blur

In my last article, we generated terrains with colors assigned based solely on elevation. Looking closely at some of the resulting terrains, you may have noticed that the colors created a layered effect. For this article, the *FractalWorld3* example uses your choice of random numbers or Perlin noise to blur the colors to eliminate the layered affect. Have a look at Figure 2 to see this example in action.

In this example, the noise function is used to blur the boundaries between colors to make the transitions less apparent. The effect is implemented by nudging the color index with the noise function as shown in part in Listing 1.

The color index is determined normally and then a delta value is calculated with the noise function. The sum of the index and the delta value is rounded and clamped to create the new color index. This method uses the row, column and elevation as arguments to the noise function. All three are scaled down to focus the noise based on trial and error. You can think of the divisors as a zoom function into the noise. Because the noise is defined in a limited-size lattice, the zoom factor focuses the range: higher zoom results in less noise range. Finding the right recipe for an effect is mostly an art but luckily others have shared their recipes.

Texturing with Noise

A popular use for noise is to generate the colors on a texture. We can apply the texture to a shape, giving the appearance of natural materials like wood or marble. Have a look at Figure 3 for an example of an image produced by the *PerlinNoiseSphere*.

Java3D supports texturing of a shape by setting the texture image on the appearance object. The *PerlinNoiseSphere* example uses a Java3D *Sphere* primitive as the shape and Perlin noise to generate the texture. The *Sphere* primitive is used in this example so some texturing details can be automatically done for us. Setting up the texture on the appearance is shown in part in Listing 2. The *getImage()* method is where the magic happens. The recipe is used to determine the noise values and the *PerlinNoiseSphere* example interprets the values as colors. Before I disclose the secret to this trick, I should mention that there's no relationship between how the recipe creates the texture and how nature creates the material. These recipes have been arrived at through trial and error and bit of luck. The results look amazingly close to the real thing, which teaches us that: In 3D graphics, there's nothing like a great fake.

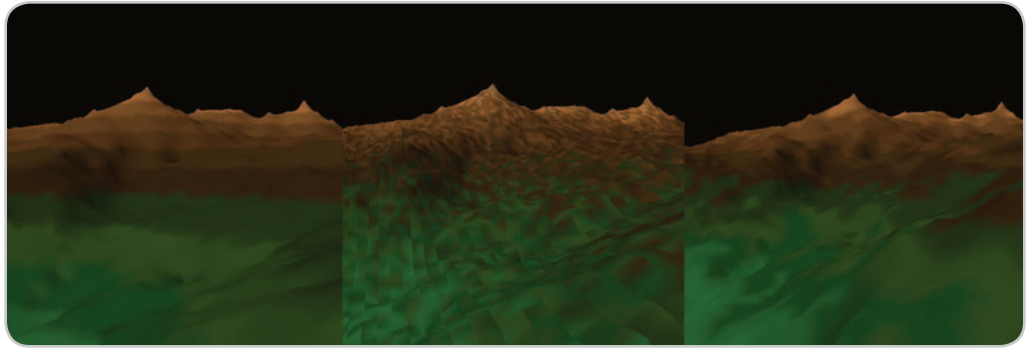


Figure 2 Blurring the color layers with random numbers and Perlin noise

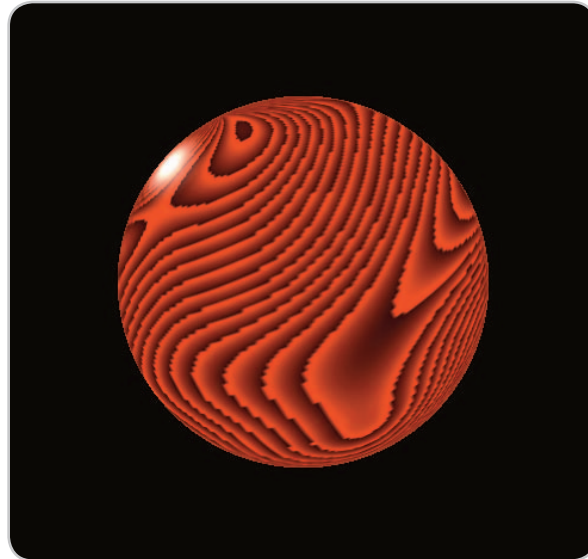


Figure 3 Perlin noise can be used to generate natural-looking materials like wood

The recipe for the wood texture in Listing 3 is decidedly simple.

The grain value is determined by the *noise* method using the image row and column as parameters. The color for the image pixel is based on the red, green and blue values calculated with the grain. Creating static texture images with noise is interesting, but the power of noise is even greater when combined with animation.

Animated Behavior

A Java3D behavior links keyboard, mouse or temporal events with changes to the scene or view. For example, the keyboard or mouse can be used to update the view allowing us to move the virtual camera through the scene. Java3D includes this support with the *KeyNavigatorBehavior*, *MouseRotate*, *MouseTranslate* and *MouseZoom* behaviors. Time can be used to animate the movement or morph the shape attributes in our scene and Java3D includes subclasses of *Interpolator* for this as well. While there are a number of behaviors available in Java3D, it's likely you'll eventually need to create your own behavior and Java3D is designed for that.

When a behavior is created, the constructor typically defines the triggering or wake-up condition such as a keyboard or mouse event, a number of frames or the passage of time. Behaviors are added to the scene like any other Java3D object.

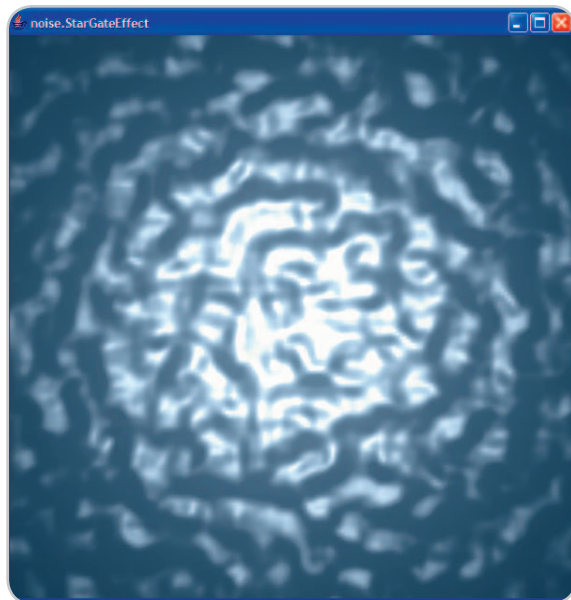


Figure 4 Noise can be used to create a Stargate effect

When your scene is initially rendered, Java3D calls the *initialize()* method where your implementation should set the trigger. When Java3D detects the triggering event, it calls the *processStimulus()* method on your behavior. Your implementation of this method does its thing and then must reset the trigger. The documentation for the *Behavior* class is excellent, so refer to it for more details.

The *ElapsedTimeBehavior* example is the basis for the animation examples in this article. When triggered, this behavior calls the *tick()* method on the configured listener after the specified number of milliseconds has passed. Milliseconds are used as the trigger rather than the number of frames so it runs consistently across different computers. Let's use this behavior to recreate the animation of a movie special effect in Java3D.

Stargate™ J3D

In 1994, Kleiser-Walczak created special effects for the movie "Stargate." One of those special effects was to create a vertical liquid doorway that would transport anything entering it across the galaxy. The original effect was done with an advanced particle system, but we can approximate it with Java3D and Perlin noise. The approach is to create a surface with fluid-like waves that rise and fall with time. The waves of the surface can be created with a height map with the height of the waves determined by a fractal function of noise. (Refer to my previous article for an overview of fractals.) Using a fractal approach creates "self-similar" ripples in the waves. The animation of waves can be accomplished by "scrolling" through the noise by changing one of the noise parameters with time. A single frame of the results is captured in Figure 4.

My first attempt to implement the surface used the *GeometryInfo* class. Many Java3D examples use this convenient utility class so many Java3D programmers depend on it. I quickly found that the general implementation was killing animation performance.

Despite attempts to reduce memory burn in my last article, the implementation of *GeometryInfo* internally makes a copy of coordinates, colors, etc. Since our approach is to change the geometry in real-time, making a copy of thousands of floating-point numbers took a lot of time. To address this, I abandoned the use of *GeometryInfo* and used a feature of *GeometryArray* called "by reference." This feature lets the original arrays be used directly eliminating copying completely.

Let's go over the highlights of moving away from the *GeometryInfo* class.

In many ways, using the geometry array classes is very similar to using the *GeometryInfo* class. One detail that the *GeometryInfo* class handles for us is called the *vertex format*. A vertex format is used to determine how much information is kept for each vertex. Examples of the vertex information include coordinates, normals, texture coordinates and colors. The vertex format is a parameter on the constructor of geometry array classes such as *IndexedTriangleStripArray*. A typical vertex format is implemented as a logical or of several integers each representing a type of vertex information as shown in Listing 4.

Note the use of *BY_REFERENCE* to indicate the direct use of our arrays. The coordinates and normals round out the vertex format for the Stargate effect. The coordinates represent the physical shape, but what about the normals? Many Java3D programmers are intimidated by the math behind normal generation. To add to this anxiety, dropping the use of *GeometryInfo* eliminates the convenience of the *NormalGenerator*. Why is this important? Java3D supports smooth shading making our hard-edged triangles look smooth. Shading requires that normal vectors be created, so now we need to generate our own normal vectors. Shading has nothing to do with shadows, but rather how light affects the color of a surface.

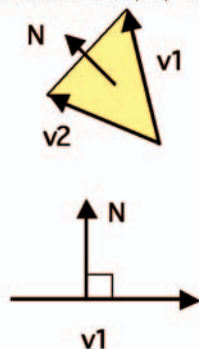
Don't Panic, It's Normal!

Java3D supports shading by varying the color intensity on triangles based on the position of the light and view. The color intensity is based on the angle of inflection of the light relative to the view and the material properties of the surface.

Facet Normal

$$N = v1 \times v2$$

`normal.cross(v1, v2)`



Vertex Normal

$$N = (N1 + N2 + N3 + N4) / 4$$

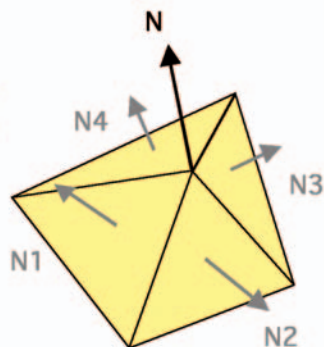


Figure 5 For smooth shading, facet normal vectors for each triangle are averaged to create a vertex normal

In other words, Java3D needs to know the direction a triangle is pointing. Mathematically, the direction of a triangle can be expressed as a unit vector that is perpendicular to the surface of the triangle. For our discussion, we will call this the *facet normal*. Taking the cross product of two vectors defined by the three points of the triangle and then normalizing the result into a unit vector determines the facet normal. The vector math package in Java3D includes extensive support for vector operations making cross-products and normalization trivial. The Java3D cross-product and ultimately the facet normal follow the “right-hand” rule. The right-hand rule is a convention for specifying the order of the two vectors in the cross-product and the direction of the resulting normal vector. (Refer to Figure 5 for an example.) We can use the vector package to compute the facet normals of each triangle. However, we need to take an additional step since shading needs something called *vertex normals*.

Vertex normal sounds contradictory, since technically you can't determine a normal for a point. Recall in my last article that Java3D can interpolate colors across a triangle using vertex colors. Since shading is based on the direction of the surface, you can think of a vertex normal as a way of specifying the direction of the surface at that vertex. Java3D calculates the color at each vertex based on the vertex normal and interpolates the color across the triangle. If all of the vertex normals of a triangle point in the same direction, the color doesn't vary across the triangle. This is called flat shading and results in faceted hard edges. To allow Java3D to smooth out the hard edges, we have to smooth the vertex normals at the shared vertices.

To accomplish this last step in the normal generation process, vertex normals are calculated by averaging the facet normals of the neighboring triangles sharing the vertex. The vertex normal for the shared vertices is set to the average value resulting in a smooth color interpolation across the neighboring triangles. This tricks your eye into thinking the surface is smooth.

Deep Below the Surface

The source code for this article includes the *Surface* abstract class that encapsulates a height map with vertex normal generation. The height map is implemented with an *Indexed-TriangleStripArray* that uses the “by reference” support. The elevations of the height map are determined by subclasses such as the *Water* class. The *Water* class uses a fractal sum of Perlin noise called fractional Brownian motion shown in Listing 5.

The result of the fractional Brownian motion is used to interpret wave heights. This recipe combined with back lighting and smooth shading creates a single frame of wavy water. We must take additional steps to enable Java3D to make those waves move. Ordinarily, Java3D optimizes the internal

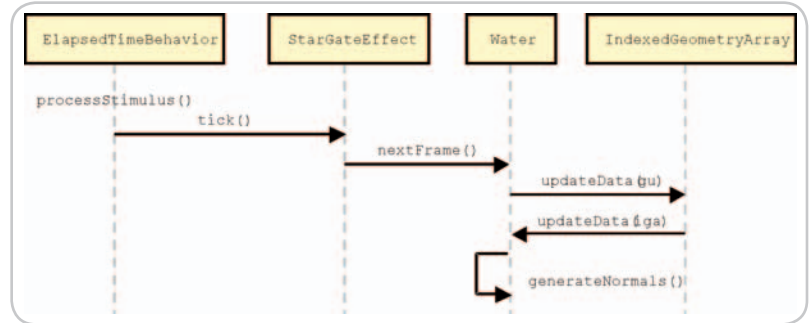


Figure 6 Updating Java3D geometry in real-time must be done via a GeometryUpdater

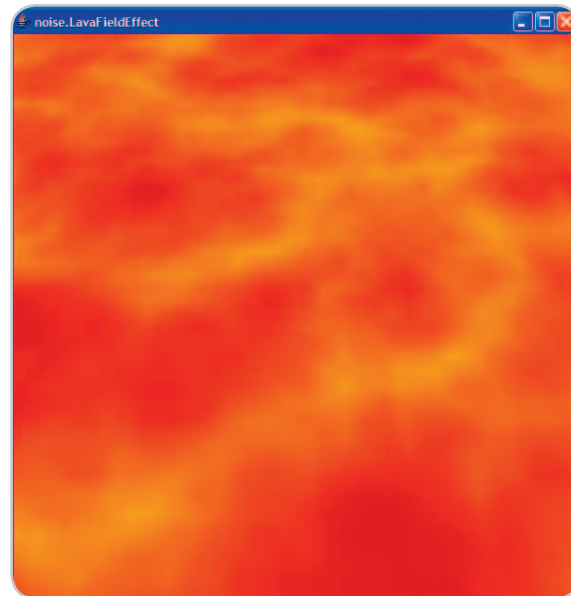


Figure 7 The turbulence noise function produces lumpy noise

representation of our shape for top performance. Before we can change the geometry of a live or compiled object, Java3D needs to know our intent. An exception is thrown if we attempt to read or write attributes without warning Java3D. Our code can express our need to access certain read and write operations by setting the *capabilities* of the object of interest. During animation of the Stargate, the example must read the geometry, change the elevations, and set the new normals. We inform Java3D of our intent by setting the capabilities in the *Water* class as shown in Listing 6.

We face one last step to complete our migration from *GeometryInfo*. Updating the geometry in real-time is complicated by the multithreaded implementation of Java3D. Activating behaviors, rendering the scene and updating geometry are done on separate threads. To update the geometry data in a thread-safe manner, we must defer the

“What if a spell could conjure roaring fires, fluffy clouds, rippling water, naturally grained wood, smooth marble and realistic terrains?”

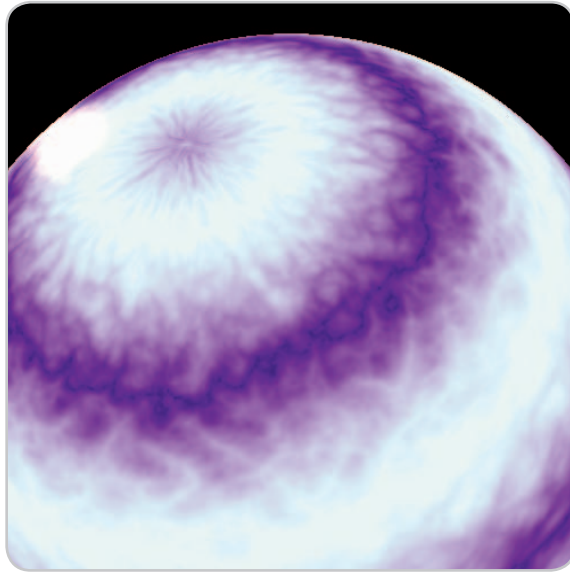


Figure 8 Turbulence can be used to create a marble texture

update to an implementer of the *GeometryUpdater* interface. To update the height of the waves, we must call the *updateData(GeometryUpdater updater)* method on the underlying *IndexedGeometryArray*. Figure 6 depicts this in a UML sequence diagram.

When Java3D triggers the *ElapsedTimeBehavior*, our implementation calls the *tick()* method on the *StarGateEffect* object. This object calls *nextFrame()* on the *Water* object, which is responsible for updating the wave heights. The *Water* object calls the *updateData(GeometryUpdater gu)* method passing itself as the *GeometryUpdater* allowing it to gain thread-safe access to the geometry and thus the referenced array data representing the elevations and normals. This approach to changing the geometry in real-time can be applied to changing textures.

Buckle Up: Turbulence Ahead

Perlin found that taking a fractal sum of the absolute value of noise created a useful recipe. The absolute value

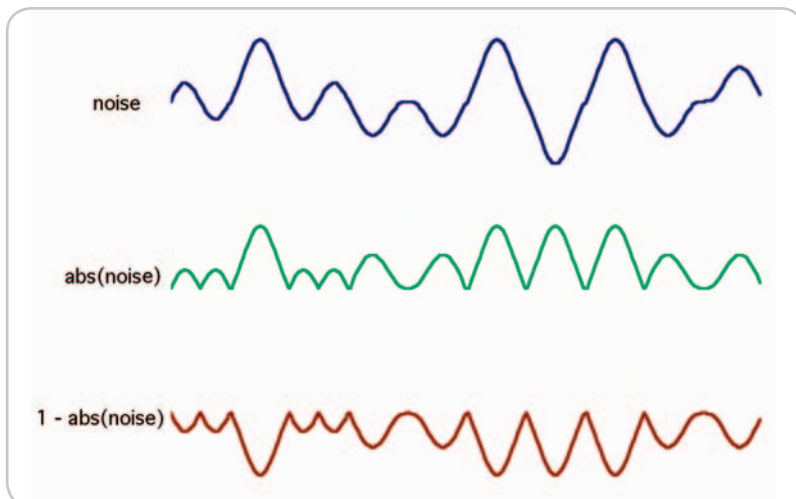


Figure 9 The absolute value of noise is turned upside down to form the ridged function used to generate terrains

of the noise creates a lumpier-looking noise curve by turning troughs into peaks. The resulting noise recipe is called *turbulence* and it's useful for creating flames, clouds and marble. The implementation of turbulence is similar to the function in Listing 4 except the absolute value of the noise is summed. If turbulence is interpreted as texture colors and animated, it looks like a smoldering field of lava. The *LavaFieldEffect* example uses turbulence to change the texture of a surface in real-time. A single frame of this example is shown in Figure 7.

The *Lava* class is a subclass of *Surface* and is responsible for updating the texture. Similar to the *Water* class, *Lava* must set the proper capabilities so the appearance and texture can be read and updated. The vertex format for the *Lava* must also include the texture coordinates option *GeometryArray.TEXTURE_COORDINATE_2*. The *Lava* class implements the *ImageComponent2D.Updater* interface so the texture can be updated in a thread-safe manner. The colors are interpreted as shown in Listing 7 using the texture column as x, the row as y and the number of frames as z.

The trigonometric sine of turbulence can be used to create a marble texture as shown in Figure 8. The *PerlinNoiseSphere* example implements an option to create the marble texture. This example isn't animated but demonstrates that useful noise recipes can be created on top of other recipes. Combining recipes is key to creating terrains with noise.

Terrains Revisited

Careers and even companies have been built with fractals and noise. F. Kenton Musgrave worked with Benoit Mandelbrot and eventually completed his doctoral thesis entitled "Methods for Realistic Landscape Imaging." Dr. Ken Musgrave is a leading authority on procedural techniques for terrain generation. He started Pandromeda, a company dedicated to creating procedural worlds through a product named *MojoWorld*. A number of the most stunning, realistic computer-generated effects for Twentieth Century Fox's movie, "The Day After Tomorrow," were created with *MojoWorld*. A noise-based terrain-generation recipe originated and shared by Dr. Musgrave is called a ridged multifractal.

The ridged function uses the absolute value of Perlin noise and turns it upside down. This approach is similar to turbulence turning the peaks of lumpy noise into troughs leaving ridges resembling mountains or sand dunes. See Figure 9 to visualize this process.

Using a fractal sum of the ridge function creates the terrain. The ridged function is called *multifractal* because the roughness of the surface varies with elevation. Unlike the midpoint displacement method described in my last article, there's no terrain size requirement to make the algorithm work. The midpoint displacement approach depends on the results of other vertices and the number of vertices on a side must be a power of two minus one. The ridge multifractal algorithm interprets elevation based on the noise parameterized with the longitude and latitude of a vertex similar to the two-dimensional noise surface in Figure 1. The elevation of each

vertex can be calculated independent of other vertices because of the inherent continuity and coherence of Perlin noise. This vertex independence makes the algorithm ideal for terrain tessellation allowing a more distant terrain to use fewer triangles than closer areas. While terrain tessellation isn't part of this article, the *FractalWorld4* example does demonstrate the ridged multifractal algorithm.

The *FractalWorld4* example uses the *Water* class and *RidgedFractalTerrain* class to generate terrains such as Figure 10. As you might guess, the *RidgedFractalTerrain* class is a subclass of *Surface*. Because the terrain isn't changed during runtime, no capabilities are set, but the vertex format includes *GeometryArray.COLOR_3* to allow for vertex coloring. The example uses noise to nudge the color index to eliminate the layering of colors. The water is added to the scene along with the terrain, and Java3D takes care of clipping the water along the shorelines for free. You can invoke the example with animated water but it can be slow since the waves are animated across the entire water surface including underground.

The End?

Perlin noise is a powerful tool limited only by your imagination. There are many other uses of noise to create even more special effects including smoke, flames, volumetric fog, clouds and even facial expressions. Just like in the movies, these will have to wait for a sequel.

Acknowledgements

I would like to thank Jeff Ryan, Scott Gerard and Al Spohn for reviewing this article. "Stargate" is a registered trademark of Metro-Goldwyn-Mayer Studios Inc. ☺

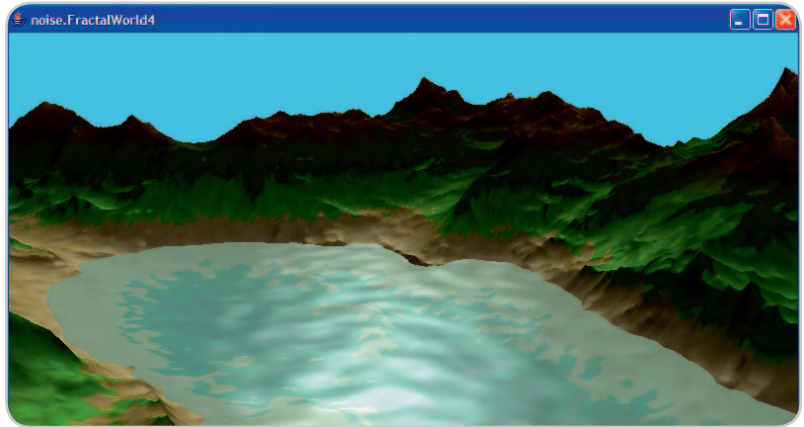


Figure 10 Noise can be used to generate terrains and water

References

- Dr. Ken Perlin's home page: <http://mrl.nyu.edu/~perlin/>
- Java reference implementation of noise: <http://mrl.nyu.edu/~perlin/noise/>
- Dr. Ken Perlin's GDC noise tutorial: <http://www.noisemachine.com/talk1/>
- "Stargate" special effects done by Kleiser-Walczak: <http://www.kwcc.com/works/ff/star.html>
- Dr. Ken Musgrave's home page: <http://www.kenmusgrave.com/>
- Pandromeda home page: <http://www.pandromeda.com/>
- "Texturing & Modeling: A Procedural Approach" by David S. Ebert, F. Kenton Musgrave, Darwyn Peachey, Ken Perlin and Steven Worley, Morgan Kaufmann Publishers, 2003.
- Ken Musgrave's doctoral dissertation: <http://www.kenmusgrave.com/dissertation.html>

Listing 1

```
// Normalize the height value to a
// color index between 0 and NUMBER_OF_COLORS - 1
float index = (NUMBER_OF_COLORS - 1)*((100f - elevation) / 100f);
float delta = 1.5f *(float)ImprovedNoise.noise(row/3.7,
                                             elevation/7.4, col/3.7);

// Nudge the color index with the noise
int answer = Math.round(index + delta);
// Clamp the index value
if(answer < 0) answer = 0;
if(answer > NUMBER_OF_COLORS - 1) answer = NUMBER_OF_COLORS - 1;
```

Listing 2

```
// Create the texture for the sphere
Texture2D texture =
new Texture2D(
    Texture2D.BASE_LEVEL,
    Texture2D.RGBA,
    IMAGE_SIZE,
    IMAGE_SIZE);
texture.setImage(0, getImage());
texture.setEnabled(true);
// Set the optional quality settings
texture.setMagFilter(Texture2D.NICEST);
texture.setMinFilter(Texture2D.NICEST);
appearance.setTexture(texture);
```

Listing 3

```
// x = column, y = row, z = static #
double noise = noise(x, y, z) * 15.0;
double grain = noise - Math.floor(noise);
int red = 71 + (int) (164.0 * grain);
int green = 34 + (int) (74.0 * grain);
int blue = 34 + (int) (24.0 * grain);
```

Listing 4

```
IndexedTriangleStripArray geometry =
new IndexedTriangleStripArray(
```

```
vertexCount,
GeometryArray.COORDINATES
| GeometryArray.NORMALS
| GeometryArray.BY_REFERENCE,
indexCount,
stripCounts);
```

Listing 5

```
static public double fBm(
    double x,
    double y,
    double z,
    int H,
    int octaves) {
    double answer = 0;
    for (int i = 0; i < octaves; i++) {
        answer = answer + noise(x, y, z) / (1 << H * i);
        x = x * 2;
        y = y * 2;
        z = z * 2;
    }
    return answer;
}
```

Listing 6

```
setCapability(Shape3D.ALLOW_GEOMETRY_READ);
Geometry geometry = getGeometry();
geometry.setCapability(GeometryArray.ALLOW_REF_DATA_READ);
geometry.setCapability(GeometryArray.ALLOW_NORMAL_WRITE);
geometry.setCapability(GeometryArray.ALLOW_REF_DATA_WRITE);
```

Listing 7

```
double turbulence = ImprovedNoise.turbulence(x, y, z, OCTAVES);
double color = Math.min(192 * turbulence, 192);
int red = 255 - (int) (0.3 * color);
int green = 192 - (int) (color);
int blue = 0;
```



DESKTOP



CORE



ENTERPRISE



HOME

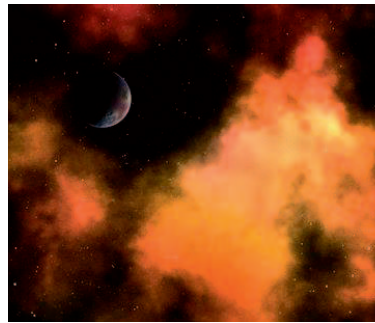
SLOOH.com Delivers Astronomy to the Mainstream

by Matt BenDaniel

SLOOH.com is the world's first and only source of live deep-sky celestial images. Every night SLOOH's telescopes scan the skies and deliver stunning images to computer screens around the world in seconds. SLOOH offers a schedule of fascinating five- and 10-minute "missions" that probe galaxies, nebulae and comets. Any SLOOH subscriber can reserve time on a telescope and direct its actions. A user needs only a 56Kb modem connection to the Internet and a Web browser. No knowledge of astronomy is necessary. Empowering novices to remotely control a professional observatory is completely unprecedented in human history.

Let's steal a look behind the scenes to understand how SLOOH's technology conquers the skies. Its operations are housed at two locations. SLOOH built a unique robotic astronomical observatory in the Canary Islands and manages enterprise-class servers at a collocation site (a.k.a. colo) in New York City.

SLOOH's unmanned observatory is connected to the outside world only through the Internet. It operates autonomously, with occasional remote maintenance over the Internet. The SLOOH observatory hardware consists of two automated domes, each with a robotic mount, a wide field imaging system and a high magnification imaging system. Each imaging system has a telescope, a



CCD astro-imaging camera, filters and a focuser. All functions of these systems are under software control running on personal computers.

SLOOH's key technological component is the custom control software running in the observatory. It is written in the Java programming language. This software:

- Handles communication with the servers
- Controls motorized actuation of the equipment
- Gathers data from instruments
- Performs image processing

The observatory's automatic image processing takes raw output from the cameras and makes stunning color images on-the-fly. Nobody – not even the Keck telescopes or NASA – has ever succeeded in doing that.

The colo installation contains the systems that handle all user interaction, scheduling and administration. A

SunFire V120 running Solaris 8 hosts an Apache 2 Web server and a MySQL 4 database. Verisign certificates are used for secure Web communications. The site makes extensive use of JSP and servlets. JRun integrates Java-based functionality and the Web site, the database and the observatory. A PC running Red Hat 8 hosts a Macromedia Flash Communication Server that powers the slick graphics of the Flash client plug-in.

Messages are sent between the colo and observatory using a custom RMI-based Remote Observatory Messaging Protocol. Every few minutes, the colo sends a command over the Internet to the observatory telling it to observe a particular celestial object. The observatory autonomously points the telescope, selects filters and focuses and photographs the object. The images are sent from the observatory to the colo. Every action of the observatory is signaled with a message. For example, when the camera shutter opens, a message is sent. The images and messages are broadcast to each and every user's browser. The browser not only displays the image; it also displays telemetry showing almost every action at the observatory.

SLOOH was designed to convey the experience of being inside a working observatory. Pre-recorded voice audio germane to the objects is available to the user, providing a personalized "astro-tour guide." ☺

Matt BenDaniel is co-founder and chief technology officer of Slooh.com. He serves as product manager, software architect and astronomer. In addition, BenDaniel designed, built and programmed Slooh's autonomous astronomical observatory. He has been a leading-edge software consultant and developer for 25 years.

matt@slooh.com

From Here to Ubiquity

—continued from page 6

Finally, *developers are a technology's strength*. The best thing you can do is to provide developers with useful tools and access to underlying code — and get out of the way.

When it was launched, Java empowered software developers to innovate and create a new vision for the Web. Now that open source development has become mainstream, a new period of software innovation has arrived, where the best technologies (not just the best-marketed ones) can actually win. And there's no going back.

Ten years later, and looking at the decade ahead, the future of software looks bright indeed. ☺

Remember the good old days?



(800)-876-3101

Ah, yes. Those were simpler times. But you're not nostalgic for old technology. If your Java application relies on SQL Server data, your database driver should give you the performance, reliability, and functionality Microsoft® SQL Server deserves. **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.

Don't use 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.