

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

JDJ

WWW.SYS-CON.COM/JDJ VOL.10 ISSUE:4

- FEATURE -

Generic Request Response Broker for JMS

PAGE 12



TAMING TEAM

CONFLICTS IN JAVA DEVELOPMENT

- Control your code -

RETAILERS PLEASE DISPLAY
UNTIL JUNE 30, 2005

\$5.99US \$6.99CAN

05>



PLUS...

▶ Delegates Reloaded:
Walking the Path

▶ Simplify Pattern
Matching

▶ Performance in
J2SE 5.0

▶ Bytecode Generation
Tips and Tricks

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

Who'll Be Getting Hitched Next?



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 Editor: Seta Papazian
Online Editor: Martin Wezdecki
Research Editor: Bahadır Karuv, PhD

Writers in This Issue

Anant Athale, Calvin Austin, Alan Barclay, Michael Birken, David Castro, Osvaldo Pinali Doederlein, Brian Duff, Yakov Fain, Jeremy Geelan, Murali Kaundinya, Lev Kochubeevsky, Sunil Mathew, Sunil Venkayala, Pavel Vlasov, Aaron Williams, Joe Winchester

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.



When in October of last year I asked the rhetorical question "Is Mergermania Back?" (JDJ, Vol. 9, issue 10), there wasn't much doubt that it already was, but it took until last month to truly demonstrate just to what extent. It's not just back; in March we saw it's back with a vengeance.

IBM has gobbled up Ascential (\$1.1 billion) and – SAP's attempts to do the same notwithstanding – Oracle just snagged Retek (\$643.3 million), a much smaller deal than its January takeover of PeopleSoft (\$10.6 billion), but a significant one nonetheless.

Its industry significance was underlined by the CEO of Retek, Marty Leestma, who was crystal clear why ERP arch-rivals SAP and Oracle had both pursued his company so keenly, and said so: "This is the first time in 15 years that there hasn't been a new game-changing technology out there, and that's driving this consolidation," he said.

In Leestma's view, everyone in the industry struggles with how to grow when there's no push coming from new technology. In the case of a company like Oracle, it clearly has the opportunity to leverage its very high profit margins and fund growth-by-acquisition using its cash hoards. So where Retek has just gone, who's to say Lawson Software, Siebel Systems, or BEA Systems won't shortly follow?

Question one for April then is: *Who'll be getting hitched next?*

After that comes the inevitable question two: *Which mergers/takeovers will be the first to be unpicked?* Who'll be the first to the divorce court? At the time of my original "Is Mergermania Back?" commentary, you may recall, Dr. Adam Kolowa, CEO of Parasoft, sounded a word of caution, asking us to remember Compaq buying DEC or HP buying Compaq – successive takeovers that Kolowa felt had both failed. "They brought more headaches than they were worth to the companies that were involved in them," he declared.

Which in turn begs the obvious third question: *Who's right?* The giants like HP or Oracle who seek to grow ever bigger (or SAP, who surely won't let their defeat by Oracle over Retek blunt their enthusiasm for some other quarry sometime soon); or those who remain independent, the way that – say – Sun seems determined to do?

Back in October, JBoss CEO Marc Fleury commented: "Consolidation is a natural fact in the software markets" and went on to note that the mergers and consolidations might well be catalyzed by the recent rise of open source software. "Open source accelerates this natural consolidation by putting great pressure on the profits of infrastructure software players," Fleury commented.

Like Leestma then, Fleury sees mergermania not so much in terms of winners and losers. Rather it's a function of efficiencies at play.

But what about Leestma's observation that "there's no push coming from new technology"? Is he right on the money there? Or is there still life in the old *i*-technology dog yet, with innovations such as Flash video for example about to rip in 2005 through the WWW – an almost classic "disruptive technology" in the precisely the game-changing sense Leestma believes is absent from enterprise software?

Disruptive technologies are usually introduced to the market by small start-up enterprises, not giants like Oracle and SAP. So an agile player like Macromedia can potentially run rings round the industry, and Clay Christensen's predictions may come very true, with Flash video gaining a strategic foothold that allows Macromedia over time to move up-market through performance improvements until it finally displaces all market incumbents and becomes the sole purveyor of quite simply the only way to deliver streaming video over the Web, just as we have been demonstrating recently at SYS-CON.TV.

Back to mergermania though. At this writing, telco rivals Qwest and Verizon both still want to conjoin with MCI, an \$8.5 billion or a \$6.7 billion deal depending on who wins – a little smaller than Oracle-PeopleSoft, but even so. Which suggests that April will turn out to be every bit as lively as March has been – perhaps even more so, if the focus moves from telecommunications back to enterprise software.

We can all confidently expect substantial ongoing M&A activity then. It's only really a question now of *when?* and *who?* Get ready for the next wave of announcements! ☺

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

Bond disparate data

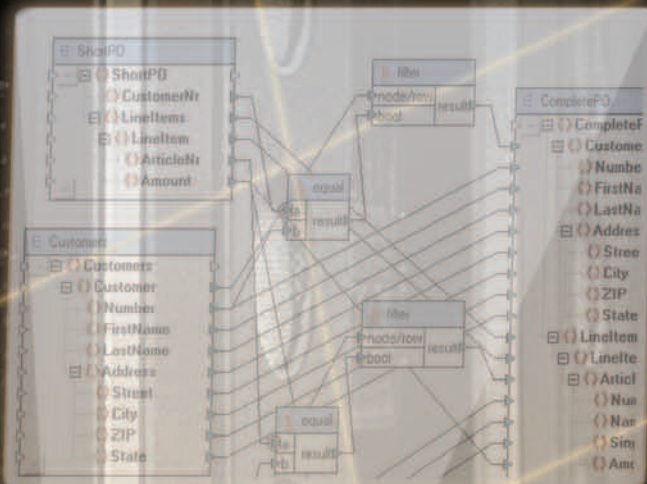
Connect with MapForce™ 2005,
the award-winning visual data
integration tool from Altova®.

Integrated in Version 2005 Release 3:

- EDI data output
- ANSI/ASC X.12 support
- Enhanced visual function builder
- Mixed content mapping

Discover just how easy MapForce 2005 makes it to exchange data between XML, database, flat file and EDI formats. Simply drag connecting lines from information source(s) to target(s) and drop in data processing functions. MapForce 2005 converts data on-the-fly and auto-generates data conversion code in XSLT 1.0, XSLT 2.0, XQuery, Java, C++, or C# for royalty-free use in custom data integration applications. Give your data direction!

Download MapForce™ 2005 today: www.altova.com



JDJ contents

JDJ Cover Story

32

Taming Team Conflicts in Java Development

– Control your code –

by Brian Duff



FROM THE GROUP PUBLISHER

Who'll Be Getting Hitched Next?

by Jeremy Geelan

3

JAVA ENTERPRISE VIEWPOINT

Our JUGs Need a Push-Up

by Yakov Fain

6

SOLUTIONS

Bytecode Generation Tips & Tricks

Simplify runtime

by Pavel Vlasov

8

CMS

Enterprise Content Management on the Java Platform

A peek into Java standards APIs for accessing a content repository

by Murali Kaundinya and Sunil Mathew

18

ANALYSIS

Using Java Data Mining to Develop Advanced Analytics Applications

The predictive capabilities of enterprise Java apps

by Sunil Venkayala

24

CORE AND INTERNALS VIEWPOINT

Java: What Does the Future Hold?

by Calvin Austin

30

REGULAR EXPRESSIONS

Simplify Pattern Matching

Use java.util.regex

by Anant Athale

36

NEW FEATURES

Performance in J2SE 5.0

What's new and what's faster

by Osvaldo Pinali Doederlein

42

DESKTOP JAVA VIEWPOINT

Geeks, Germs, and Software

by Joe Winchester

46

LABS

dotJ JSP Tag Library 2.0.7

Master of tags

Reviewed by David Castro

54

LABS

Putting Energy Software Tools to the Test

From set top boxes to wireless messaging

Reviewed by Alan Barclay

56

JSR WATCH

Evolving the JCP Program Within the Java Ecosystem

Keeping the community operating effectively

by Aaron Williams

62

Features

12



Generic Request Response Broker for JMS

by Lev Kochubeevsky

48



Delegates Reloaded: Walking the Path

by Michael Birken



Yakov Fain
Contributing Editor

Our JUGs Need a Push-Up

TIt's been almost 10 years since enthusiasts around the world started to form small local communities called Java Users Groups (JUGs). They gather once in a while after work to network, listen to a presentation on some new Java technology or JSR, and talk about what's hot and what's not.

I'm a member of two JUGs: New York City (takes place at Sun Microsystems' office, invites well-known speakers, offers free food and gives away trinkets like plastic water containers and T-shirts) and New Jersey (a room with a projector in a small township's rescue squad, no food, and the speakers are Java junkies like you and me). Although they have such different sponsors, both groups meet religiously once a month and their leaders deserve credits for this part-time job that does not offer any monetary rewards.

Sun Microsystems maintains a Web site <http://java.sun.com/jugs/> devoted to JUGs around the world. And you can find a JUG list at <https://jugs.dev.java.net/>. But I decided to do my own little Google research and here's my personal take on the state of the JUGs. I looked for JUGs that have a real place to meet, not pure Internet forums.

North America: USA is a clear winner. There are dozens of groups, and there are JUGs that meet on a regular basis in New York City, San Diego, Silicon Valley, Dallas and Philadelphia.

Europe: London and Saint Petersburg are real and active.

South America: Go, Brazil, go! It has multiple JUGs. Even though football and Samba are still more popular there, Java is catching up.

Asia: Small Hong Kong represents almost half the earth's population. What happened to India and mainland China?

Australia: Wake up, guys! You're not really that far from the rest of the world!

I found some interesting PowerPoint slides and code samples from past presentations on the Web sites of these JUGs.

Why JUGs Are Good for You

For Java programmers: A great way to keep up with new Java technologies. It also gives you a chance to network with your peers, which always helps in getting a job.

For business: Instead of paying headhunters hefty finder fees every time you need to hire a Java developer, create a home for your local JUG by letting them use one of your conference rooms one evening a month. The return on this small investment will be phenomenal: whenever you need a Java person, post your job requirements to the JUG's mailing list.

For students: JUGs give you a chance to learn what's happening in the real business

world and maybe find an internship or summer job.

For headhunters: Your local JUG may give you an access to a pool of Java developers. But you should contribute too: speak periodically on the status of the Java job market in your geographical area. Such

sessions are pretty popular with developers. Here's a tip for you. Headhunters can create new JUGs and breed Java programmers for themselves.

For authors: It's a good way to promote your books. Don't be shy though; always give away a free copy of your book.

How to Start a New JUG

If you can't find a local JUG on Sun's web site, no worry. Create one by following these steps.

1. Start a new group using a free service from Yahoo at groups.yahoo.com. It'll take you less than 15 minutes to set up a place online where all members of your group can post messages (moderated, if needed), upload files, schedule meetings, and vote. I've been using this service for years and it works like charm.
2. Compile a list of all the Java developers you know and invite them to join this new group. In the same invitation ask them if they know of any firm, college, or a place of worship that may be willing to host your once-a-month meeting.



Yakov Fain works as a Java architect for a major bank in New York City. He wrote the book *The Java Tutorial for the Real World*; an e-book *Java Programming for Kids, Parents and Grandparents*; and several chapters for the book *Java 2 Enterprise Edition 1.4 Bible*. Yakov holds a masters degree in applied mathematics. For more information please visit www.smartdata-processing.com.
yakovfain@sys-con.com

President and CEO:
Fuat Kircaali fuat@sys-con.com
Vice President, Business Development:
Grisha Davida grisha@sys-con.com
Group Publisher:
Jeremy Geelan jeremy@sys-con.com

Advertising

Senior Vice President, Sales and Marketing:
Carmen Gonzalez carmen@sys-con.com
Vice President, Sales and Marketing:
Miles Silverman miles@sys-con.com
Advertising Sales Director:
Robyn Forma robyn@sys-con.com
National Sales and Marketing Manager:
Dennis Leavey dennis@sys-con.com
Advertising Sales Managers:
Megan Mussa megan@sys-con.com
Kristin Kuhnle kristin@sys-con.com
Associate Sales Managers:
Dorothy Gil dorothy@sys-con.com
Kim Hughes kim@sys-con.com

Editorial

Executive Editor:
Nancy Valentine nancy@sys-con.com
Associate 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
Video Production:
Frank Moricco frank@sys-con.com

Web Services

Information Systems Consultant:
Robert Diamond robert@sys-con.com
Web Designers:
Stephen Kilmurray stephen@sys-con.com
Percy Yip percy@sys-con.com

Accounting

Financial Analyst:
Joan LaRose joan@sys-con.com
Accounts Payable:
Betty White betty@sys-con.com
Accounts Receivable:
Gail Naples gailn@sys-con.com

SYS-CON Events

President, SYS-CON Events:
Grisha Davida grisha@sys-con.com
National Sales Manager:
Jim Hanchrow jimh@sys-con.com

Customer Relations

Circulation Service Coordinators:
Edna Earle Russell edna@sys-con.com
Linda Lipton linda@sys-con.com
Monique Floyd monique@sys-con.com
JDJ Store Manager:
Brunilda Staropoli bruni@sys-con.com

—continued on page 10

Parasoft **Automated** Unit Testing, Leak Detection & Best Practices:

Specifically designed for under-staffed, over-committed development organizations like yours.



Deliver better, safer software... In less time. With fewer resources.

Automate team-wide testing and best practices with Insure++, C++Test, Jtest, and other Parasoft products.

Parasoft products automatically ensure that your entire development organization is meeting requirements and adhering to a common standard – enabling your team to create high quality software without added effort or workload. Our products quickly identify security violations, memory errors, runtime exceptions, common defect patterns, and violations of best coding practices for all major development environments (including C/C++, Java, Web Services, .NET and more).

Increase development effectiveness by allowing:

- Developers to correct problems early and reduce time spent chasing bugs
- Architects to ensure that development is adhering to a common standard
- Managers to improve project visibility, decision making and predictability

Learn how leading development teams are using Parasoft products to improve the quality and efficiency of their processes. Go to www.parasoft.com/Products



Automated Software Error Prevention™

Bytecode Generation Tips and Tricks

by Pavel Vlasov

Simplify runtime

This article introduces readers to bytecode generation and shows how to inject generated bytecode into a JVM runtime. After reading this article generating a Java class won't be any harder than creating an XML document with the DOM API.

Over last couple of years, bytecode generation has gained significant momentum. Many tools generate bytecode instead of source code to obviate the compilation step and simplify the injection of generated code at runtime.

There's a number of bytecode-generation libraries with BCEL (Byte Code Generation Library) being the most renown and probably most powerful. It's used by tools such as Xalan's stylesheet compiler and Mercury Interactive's Topaz J2EE Probe.

Bytecode vs Source Generation

You may ask: "Why bother with bytecode generation if I can produce Java source?" Ultimately, it's your call what technique to use. For example, some people create XML documents with `println()`, others use the DOM API. Keep the following considerations in mind:

- With Jad, Jadclipse or DJ Java Decompiler (see Resources) a .class file is an open book as .java file. But read-only. So there's no need in (and actually no place for) formidable headers like "Generated by XXX – Do not edit!"
- There is an additional compilation step if you produce source instead of bytecode. Compilation at runtime is a real big headache, especially if you don't control the target JVM.

- The amount of Java code you have to write to produce Java source won't be significantly less than the amount of Java code for bytecode generation. Actually, it may be more.
- Using different template engines has its own vices. First of all generation logic gets distributed between the Java code and template code, which makes the whole solution less manageable and more fragile, because templates aren't easy to debug. Second, as you start adding more and more to your templates they will soon become unreadable. Take a look at Xdoclet templates – they look like cuneiform on the Hammurabi stella.

Use Scenarios

BCEL can be used for code generation (this is what XSLTC and SQLC do) and for code modification (this is what the Mercury Topaz J2EE Probe does). This article covers the first scenario – code generation. In MDA parlance it describes how to build the *T* in *PIM+T* -> *PSM* equation where *PSM* is Java

bytecode and *PIM* is a XSL stylesheet in the case of XSLTC and SQL statements in the case of SQLC.

Truth to tell BCEL API is a bit cumbersome...*elaborate*... I've created several helper classes to make life easier.

Runtime Generation

Classes can be generated at build time and runtime. The first case is a trivial one – generated classes can be used like any other Java classes.

Runtime code generation is a more interesting theme. When you generate classes at runtime you need to make them available to JVM. This can be done with *com.pavelvlasov.InjectingClassLoader*:

```
1.Interface myInterface=
2.new Interface(
3. "public interface com.myorg.myapp.
MyGeneratedInterface extends java.
io.Serializable",
4. "My Generated interface",
5. null);
6....
7.// Injecting
8.ClassLoader parentClassLoader = ...
9.InjectingClassLoader icl=new InjectingClass
sLoader(parentClassLoader);
10.icl.consume(myInterface);
11....
```

We injected generated classes and/or interfaces in the Java runtime. How to use them? If generated classes implement interfaces or extend classes known at compile time then the answer is obvious – instantiate and cast.

What if generated classes don't comply with the statement above? How to use classes not known at compile time? Well, the answer is that information about generated classes can be



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

vlasov@pavelvlasov.com



obtained through reflection. Scripting environments such as JSP, JSTL, Velocity, and script interpreters won't distinguish injected classes from any other.

An important note about runtime generation: BCEL isn't thread-safe. If you're going to generate classes in multithreaded environment then each generating thread should have its own classloader for BCEL classes. This will result in a bigger memory footprint because BCEL classes will be presented in memory once per generating thread.

Generating Interfaces

Generating interfaces is the simplest task because interfaces' methods are all abstract.

```
1.Interface myInterface=new
Interface("public interface com.myorg.
myapp.MyGeneratedInterface extends java.
io.Serializable", "My Generated interface",
null);
```

Adding a method is as straightforward as creating an interface itself:

```
1.myInterface.addMethod("void
setMyValue(java.lang.String str)", null,
"My generated method");
```

If the method parameters aren't known at coding time then they can be supplied in the second parameter of `addMethod()`, which will either be null or a collection of `com.pavelvlasov.util.Parameter` implementations.

Adding a field is also a one-liner:

```
1.myInterface.addField("MY_CONSTANT",
"java.lang.String");
```

But fields in interfaces are static final and shall be initialized in the static initializer:

```
1.InstructionList il=new InstructionList();
2.il.append(new LDC(myInterface.getClass-
Gen().getConstantPool().addString("My con-
stant value")));
3.il.append(myInterface.createPutField("MY_
CONSTANT"));
4.il.append(new RETURN());
5.myInterface.addStaticInitializer(il,
"Initializes myInterface");
```

Once you've created an interface and added methods you should either save it to a file for future use or inject it

into the runtime. To save the interface you can invoke its `save(File)` method or obtain BCEL `JavaClass` object using `getJavaClass()` method and do whatever you want with that object.

In case the generated interface needs to be injected into Java runtime `com.pavelvlasov.codegen.InjectingClassLoader` comes into play.

```
1.ClassLoader parentClassLoader = ...;
2.InjectingClassLoader icl=new InjectingCla-
ssLoader(parentClassLoader);
3.icl.consume(myInterface);
4.Class myInterfaceClass=icl.loadClass("com.
myorg.myapp.MyGeneratedInterface");
5....
```

Generating Classes

Generating classes is a bit more complicated than generating interfaces since you need to generate method implementations.

My advice on code generation is to:

- Minimize the amount of code to be generated by moving the functionality to the superclass and helper classes
- Create a template method in Java and compile it.
- Run `org.apache.bcel.util.Class2-HTML` to generate class HTML documentation.
- Run `org.apache.bcel.util.BCELifier`. It will create code, which will generate a template class. There's a bug in BCELifier shipped in BCEL 5.1 and it doesn't work on all classes. See Resources below for a download link to the fixed version.
- Use instructions produced by BCELifier as a starting point. You can also copy bytecode instructions from generated HTML documentation to your generator method; comment them out and then write code using commented instructions as guidelines.
- Run `org.apache.bcel.verifier.Verifier` or `com.pavelvlasov.codegen.ClassGeneratorBase.verify()` on the generated classes.
- Use Jad, Jadclipse or DJ Java Decompiler to decompile generated files and verify method logic. I recommend that you set the "annotate" option so you can see the bytecode instruction as comments.

Writing linear bytecode is very simple as we've seen from the previous section.

Branches (if, while,...) and exception handlers are the things that require attention. Let's see how to generate code that has both branches and exception handlers using the advice above. This is the code we're going to generate:

```
1.public int templateMethod(String str) {
2. if (str==null) {
3. return 0;
4. } else {
5. try {
6. return Integer.parseInt(str);
7. } catch (NumberFormatException e) {
8. return -1;
9. }
10. }
11.}
```

This is the output of BCELifier:

```
1.InstructionList il = new
InstructionList();
2.MethodGen method = new MethodGen(ACC_
PUBLIC, Type.INT, new Type[] { Type.STRING
}, new String[] { "arg0" }, "template-
Method", "com.pavelvlasov.codegen.samples.
TemplateClass", il, _cp);
3.
4.InstructionHandle ih_0 = il.append(_fac-
tory.createLoad(Type.OBJECT, 1));
5.BranchInstruction ifnonnull_1 = _factory.
createBranchInstruction(Constants.IFNONNULL,
null);
6.il.append(ifnonnull_1);
7.InstructionHandle ih_4 = il.append(new
PUSH(_cp, 0));
8.il.append(_factory.createReturn(Type.
INT));
9.InstructionHandle ih_6 = il.append(_fac-
tory.createLoad(Type.OBJECT, 1));
10.il.append(_factory.createInvoke("java.
lang.Integer", "parseInt", Type.INT,
new Type[] { Type.STRING }, Constants.
INVOKESTATIC));
11.InstructionHandle ih_10 = il.append(_
factory.createReturn(Type.INT));
12.InstructionHandle ih_11 = il.append(_
factory.createStore(Type.OBJECT, 2));
13.InstructionHandle ih_12 = il.append(new
PUSH(_cp, -1));
14.InstructionHandle ih_13 = il.append(_
factory.createReturn(Type.INT));
15.ifnonnull_1.setTarget(ih_6);
16.method.addExceptionHandler(ih_6, ih_10,
ih_11, new ObjectType("java.lang.NumberForm-
atException"));
17.method.setMaxStack();
18.method.setMaxLocals();
19._cg.addMethod(method.getMethod());
20.il.dispose();
```

Now, we change it to be less cryptic:

```
1.MethodPrototype mp=new
MethodPrototype(myClass, "public int
getMyInt(java.lang.String str)", null);
2.InstructionList il = new
InstructionList();
3.ExceptionHandler eh=new
ExceptionHandler("java.lang.
NumberFormatException");
4.il.append(mp.createVariableLoad("str"));
5.BranchInstruction ifnonnull =
InstructionFactory.createBranchInstruction(Co
nstants.IFNONNULL, null);
6.il.append(ifnonnull);
7.il.append(new ICONST(0));
8.il.append(mp.createReturn());
9.InstructionHandle ih = il.append(mp.create
VariableLoad("str"));
10.ifnonnull.setTarget(ih);
11.eh.setFrom(ih);
12.il.append(myClass.createInvoke("java.lang.
Integer", "int parseInt(java.lang.String)",
null, Constants.INVOKESTATIC));
13.eh.setTo(il.append(mp.createReturn()));
14.eh.setHandler(il.append(InstructionFactory
.createStore(Type.OBJECT, 2)));
15.il.append(new ICONST(-1));
16.il.append(mp.createReturn());
17.Collection ehc=new ArrayList();
18.ehc.add(eh);
19.mp.addMethod(il, ehc, "My generated
method");
```

After that we run *org.apache.bcel.verifier.Verifier* to verify the generated class. Then we run Jad and compare the decompiled code with the original. Once you become familiar with bytecode, you won't need BCELifier. I use only Jadclipse with annotations.

Generating Documentation

If you generate classes that imple-

ment some interface then you probably don't need to document them. For example, Xalan XSLTC produces a bunch of classes but you don't need documentation for these classes, all you need to know is the class name. On the other hand, SQLC (see the Resources) generates classes and interfaces from SQL statements and in this case documentation is necessary. The good thing about classes from the *com.pavelvlasov.codegen* package is that if you use them to generate bytecode then the work to generate documentation is close to zero. You need to do is to use the class *com.pavelvlasov.codegen.HtmlDocConsumer*. You can also use *xml.DocConsumer* to produce XML documentation and then style it. Stylesheets are available in the SQLC sample application.

```
1.HtmlDocConsumer consumer=
2.new HtmlDocConsumer(
3.new File("generated"),
4.new File("generated_doc"));
5.
6.com.pavelvlasov.codegen.Class myClass=
7.new com.pavelvlasov.codegen.Class(
8."public class com.myorg.myapp.
MyGeneratedClass",
9."My Generated class",
10.consumer.getListener());
11....
12.// Saving to file.
13.consumer.consume(myClass.getJavaClass());
14.consumer.shutdown();
```

Conclusion

I hope that after reading this article bytecode generation will become part of your skillset.

There are many cases when there's a model/data structure in a non-Java software system that should be exposed

to Java. It can be an XML schema, a mainframe map, database metadata... There's probably no reason to write a generator for an XML schema because it's already done many times, just select an XML-Java binding solution that fits your needs. But for special cases bytecode engineering is a good way to generate bridges. ☘

Resources

- Source code: www.pavelvlasov.com/articles/bcg/source.zip. You need to download the PvCommons library (see below) to run the samples.
- BCEL: <http://jakarta.apache.org/bcel/>
- BCEL with fixed bug: www.pavelvlasov.com/bcel-5.1-fixed.zip
- JVM instructions reference: <http://cat.nyu.edu/~meyer/jvmref/>
- Antlr: (wwwantlr.org) – Parser generator.
- Jad (<http://kpdl.tripod.com/jad.html>) – fast Java decompiler
- Jadclipse (<http://sourceforge.net/projects/jadclipse/>) - Eclipse plugin for Jad
- DJ Java Decompiler (<http://members.fortunecity.com/neshkov/dj.html>) – GUI for Jad
- PvCommons (<http://www.pavelvlasov.com/pv/content/Products/Common/products.common.html>) – my library of common classes
 - *com.pavelvlasov.codegen* package – Code generation classes.
 - *com.pavelvlasov.sqlc* package – SQL compiler. Uses codegen classes to compile SQL statements to Java classes.
 - *com.pavelvlasov.cache.sql* – Example of classes compiled by SQLC

Our JUGs Need a Push-Up

—continued from page 6

3. Talk to the human resources department where you work, explain the potential benefits to them and ask for a conference room.
4. Pick up an interesting Java technology and prepare the first presentation.
5. Create a simple Web site for your new Java group that will contain information on your future and past presentations, a link to the Yahoo group you've created, and directions to where you're meeting next.
6. If you know of any Java authors, enterprise architects, or business technical leaders in your area, invite them to speak in your group. You'll be surprised, but most of them will accept your invitation and do it for free.

Even if you don't have any distinguished software engineers in your area, many programmers have worked on some interesting projects and technologies and can rattle off a short case study. Here's another idea that will keep your new group busy for a year: take the famous Pet Store application and ask the members to create and present 12 different versions of it using various Java technologies.

As the old saying goes, "If there's a will, there is a way." The JUG movement needs a little push and it's a good opportunity for you to make a difference. ☘

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

Generic Request Response Broker for JMS

A robust and easy-to-use implementation of the Request-Response paradigm

by Lev Kochubeevsky

This article describes the design and implementation of a generic request/response broker (RRB) for JMS. RRB augments JMS with a highly efficient implementation of the request-response paradigm.

It's extremely important for many asynchronous applications since JMS has somewhat weak built-in support for the request-response paradigm that might be inadequate for high-load enterprise applications. RRB solves the following common problems. It:

- Provides a generic high-performance application and JMS provider-independent implementation of the request-response pattern.
- Supports timeouts for non-blocking clients.
- Allows efficient handling of a large number of requests/responses.
- Lets one hide all the JMS details behind the façade interface.
- Reduces performance degradation when a request travels through multiple façade proxies.

Possible Solutions and Their Performance Comparison

Request-response pattern is popular in enterprise applications: a client application sends a request message to a server app, which handles the request and sends the response back to the client. Here's what JMS offers for this pattern:

- JMSMessageID and JMSCorrelationID headers can be used to link requests and responses.
- JMSReplyTo header lets the recipient of a request know where to send the response.
- TopicRequestor/QueueRequestor helpers allow sending a request and block until a response is received.

TopicRequestor/QueueRequestor is the only JMS built-in "request-response broker" and has three drawbacks:

- It's a blocking call.
- You're not allowed to specify a timeout in the only existing request() method, so your thread may be blocked forever if there's no response.

- It creates and deletes a temporary response destination for every request, which has big performance implications as we'll see later.

Because of these drawbacks, the burden of implementing timeouts and retrieving response messages efficiently is left to the developers of the client applications and that's exactly what RRB is designed to solve (The source code for this article can be downloaded from www.sys-con.com/java/sourcec.cfm.)

Timeout for Non-blocking Clients

Standard JMS lets the client provide a timeout only for the blocking (synchronous) calls in the method `MessageConsumer.receive()`: a client is blocked until the message is received or the time specified has expired. Of course, a blocking call isn't a preferred way in JMS to get a message. It's better to use a `MessageListener` object registered with a particular destination that's called whenever a message is received. Unfortunately standard JMS makes no provision for using timeouts with asynchronous calls using listeners: `MessageListener` only has `onMessage()` callback methods and no `onTimeout()`. That means your listener will be called only when a message is posted to the destination you're listening on; it'll never be called if for some reason the message you expected never arrived. In many real-life apps and in almost every request-response scenario you can't wait for a message indefinitely (especially if it's a response). You need to bail out after some timeout.

Efficient Implementation of the Response Handling

In JMS there are three ways to get a response message: 1. Create/delete a temporary destination per request

- **Pros:**
 - Clean design, response destination isn't shared, no filtering required
 - Relatively easy to implement
- **Cons:**
 - Inherently bad performance (temporary destinations are always created inside the MOM provider, which generally means two extra network calls: create and delete a temporary destination plus all the housekeeping and resources on the app server to maintain the temp destination).



Lev Kochubeevsky is a chief architect at AOL in Dulles, Virginia. His areas of interest include high-performance enterprise systems, distributed software architectures, messaging, virtual machines, simulators, and debuggers. He has a master's degree in computer science.

lev@smartneighborhood.net

2. Use the same destination for multiple responses in combination with JMS selectors for filtering
 - **Pros:**
 - Relatively clean design, no filtering in the client once the selector is set up
 - **Cons:**
 - Questionable performance: some vendors don't recommend using selectors
 - Many possible pitfalls (for example, selectors on strings are much slower than on integers)
3. Use the same destination for multiple responses without JMS selectors
 - **Pros:**
 - Can be fast depending on the implementation
 - **Cons:**
 - Filtering has to be done in the client's code
 - Many possible pitfalls that will result in lost or doubly handled messages (for example, when two or more apps use the same "unique" application_id properties for filtering)
 - High load on the client listeners (in case of a response topic) that are called every time a message is received and subsequently discarded by all but one listener

To fully understand the performance impact of the different retrieval options mentioned above I wrote a simple JMS server app (EchoClient.java) and three different JMS client apps for every option: EchoClientMultTempQs.java, EchoClientSelector.java, and EchoClient.java. These clients will show the performance metrics of each message retrieval approach. All the tests were run in the same environment: one physical Linux box using three separate JVMs, echo client, JMS provider, and an echo server.

Option 1. A new temp destination for every response:

```
java EchoClientMultTempQs echo_server.properties 1000
...
N of reqs/resps = 1000
Total time = 13765 millisecs
Average roundtrip time = 13.765 millisecs
```

Option 2. The same destination using jms selectors for filtering:

```
java EchoClientSelector echo_server.properties 1000
...
N of reqs/resps = 1000
Total time = 10705 millisecs
Average roundtrip time = 10.705 millisecs
```

As you see Option 2 is about 30% faster than Option 1 (the selector performance depends on the selector's complexity, types being used, etc.). Note that I used an integer ReqID property instead of JMSMessageID/JMSCorrelationID pair, and here's why:

1. Using an integer in a selector instead of a string improves selector performance
2. Generation of a unique JMSMessageID can be disabled for a better performance
3. JMSMessageID is a pretty long string, which makes it a bad fit if you have a JMS<->LMS (Legacy Messaging System) bridge to communicate with legacy messaging systems that usually use numeric message ids

Option 3. The same temp destination for multiple responses, custom filtering:

```
java EchoClient echo_server.properties 1000
N of reqs/resps = 1000
Total time = 5286 millisecs
Average roundtrip time = 5.286 millisecs
```

The last client (the same temporary queue for multiple responses) offers far superior performance, the only problem is it's not that easy to implement, because every application needs to have some broker layer that would create a temporary destination, listen on this destination, retrieve the response message from the response destination, figure out which listener object is waiting for this response, and call this listener. This layer is application-independent from both the client and the business service parts so you can think of it as a "Service Delegate" for JMS that's analogous to the "Business Delegate" as it's described in "Core J2EE Patterns," 2nd Edition, Prentice Hall, 2003. The difference is that it's not business-specific, that's why I call it a "Service Delegate."

Let's formulate the major features Request Response Broker should provide:

1. Temporary Response Destination functionality (creation and retrieval of response messages)
2. Efficient mechanism to record multiple outstanding request states and map incoming responses to the correct listeners, completely relieving the client from doing any filtering
3. Providing timeout functionality: client is guaranteed to be called either with the result or with a timeout
4. Efficient implementation of the timeout mechanism that recycles timeout threads (ideally uses just one timeout thread).

RRB Design and Implementation

Let's start with a generic callback interface for the RRB clients. RRB will use this interface to call a client back with a response or timeout (see Figure 1).

Callable is just a marker interface and it's purpose as well as JMSCallable and RemotelyCallable will be explained later. Let's look at the LocallyCallable interface (see Listing 1).

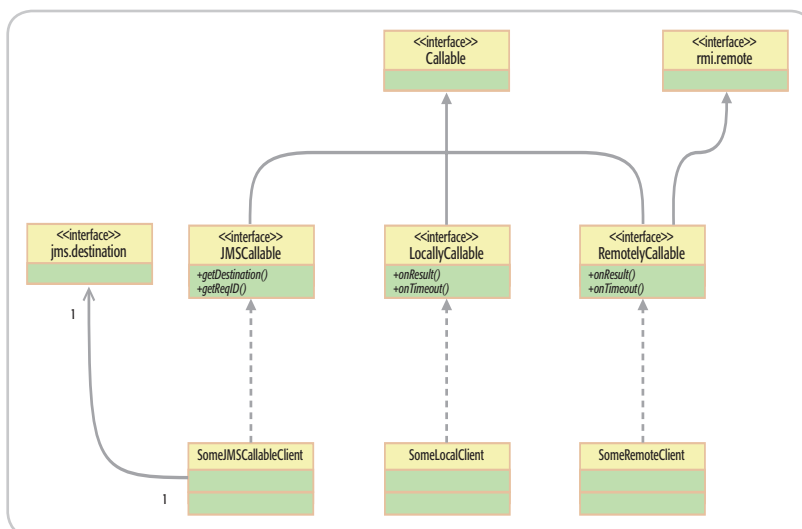


Figure 1 RRB Callback Interfaces

I had to answer the following questions while designing this interface:

- Should it extend `MessageListener` and just add `onTimeout` to the existing `onMessage`? Here's my reasoning for not doing this:
 - I wanted to have a generic callable interface that could be used outside JMS as well.
 - I wanted to have a clear distinction (enforced by a compiler) between `MessageListener` and `LocallyCallable` so it would be impossible to use the `LocallyCallable` implementation as the `MessageListener` in the regular JMS API. I wanted to avoid the situation where a programmer switches from RRB to regular JMS API and still expects `onTimeout` to be called.
- Should callback functions have a state returned as a parameter?
 - In general, the object that implements `LocallyCallable` should have all the state needed to handle the response, i.e., one callback object per request (callback object pooling recommended). In this case the state parameter should never be used (it should be null). I added it mostly for the completeness of the API and to support some applications that use the same callback object for multiple outstanding requests.

Now let's take a look at the main RRB class diagram (see Figure 2).

The major components of RRB are:

- `TimedObject` (`RRB.TimedObject.java`) is a generic abstract representation of any object that's "timed," i.e., something should be done after the timeout has expired. It can use an external `Timer` (no thread is created) or an internal one that creates a `Timer` thread per timed object. Note that RRB always uses an external timer thread that is owned by `ResponseBroker`, so there's only one timer thread regardless of the number of outstanding requests.

- `RequestState` (`RRB.RequestState.java`) holds all the information about the pending request, including a reference to the callback object and the requested timeout. It extends `TimedObject` and provides callback functionality to return either result or the timeout.
- `ResponseBroker` (`RRB.ResponseBroker.java`) is the core class of the system that provides core functionality. It creates and manages a JMS temporary queue and has the `LinkedHashMap` of outstanding `RequestState` objects, a `Timer`, and a `ResponseListener` to retrieve the incoming responses. It uses a special integer JMS property to link requests and responses, whose name is supplied by the `RRBFactory` (see below), and held in the `m_idprop` name member variable. Integer property is used for performance reasons; every destination server in the system will assign a value to it. `RequestResponseBroker` adds the request functionality and serves as an API layer for `ResponseBroker`. It implements `generateReqID` (the request id in the system doesn't have to be unique if you use multiple RRBs: every RRB gets responses only to the requests generated by this particular RRB, because every RRB has its unique temporary queue. These are the RRB's methods:
 - `prepareRequest` (`Message m`, `Callable client`, `int timeout`, `Object clientstate`) prepares the final request message to send. RRB generates `ReqID`, adds a `ReqID` property, sets the `JMSReplyTo` header to its internal temp queue, and adds this request state to its pending requests map. After calling this method, the client sends the request to the destination using the JMS API.
 - `prepareAndSendRequest` (`Message m`, `Callable client`, `int timeout`, `Object clientstate`, `Queue dest`) is a one-stop shop; it does everything that `prepareRequest` does and sends a request.
 - `sendAndWaitForResponse` (`Message m`, `int timeout`, `Queue dest`) is a blocking call that doesn't return until either result is received or the timeout has expired. Why do I provide a blocking API in RRB? Well, some applications may benefit from it. My implementation fixes two out of the three main problems with the `TopicRequestor/QueueRequestor` pair: it provides the client with the timeout feature and it doesn't create/delete a temp response destination per request (internally it's implemented as a regular client callback object, just the calling thread is blocked until callback's `onResult` or `onTimeout` is called. See the class `RequestResponseBroker.InternalLocallyCallableImpl`).
 - `cancelRequest` (`int reqid`) lets the client cancel the outstanding request without waiting for the results or timeout: it's useful when you have a group of requests, one of them fails and you don't want to continue.
- `RRBFactory` is responsible for creating and caching RRB instances. You can have multiple RRBs in the same JVM. RRB can serve one or multiple applications depending on the load and logical grouping. RRB is a pretty lightweight component that can easily handle hundreds or even thousands of outstanding requests so it doesn't make sense to have more than a handful of RRB instances. RRB is completely thread-safe and the same RRB object can be called by multiple threads simultaneously.

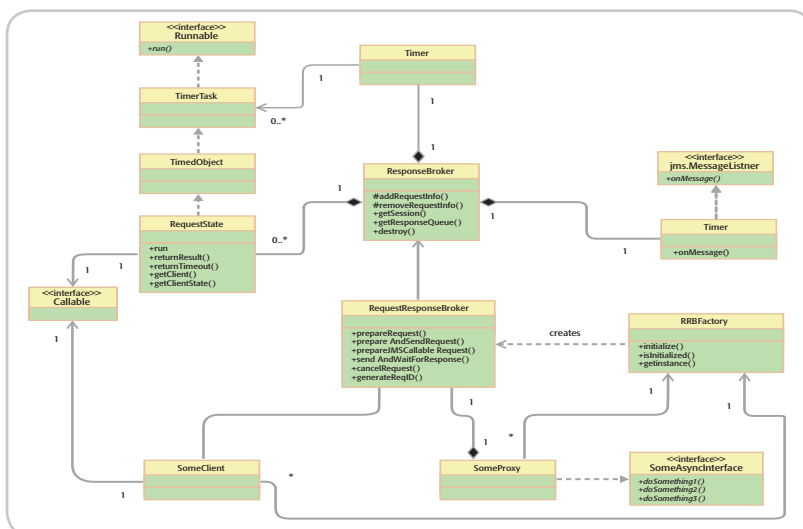
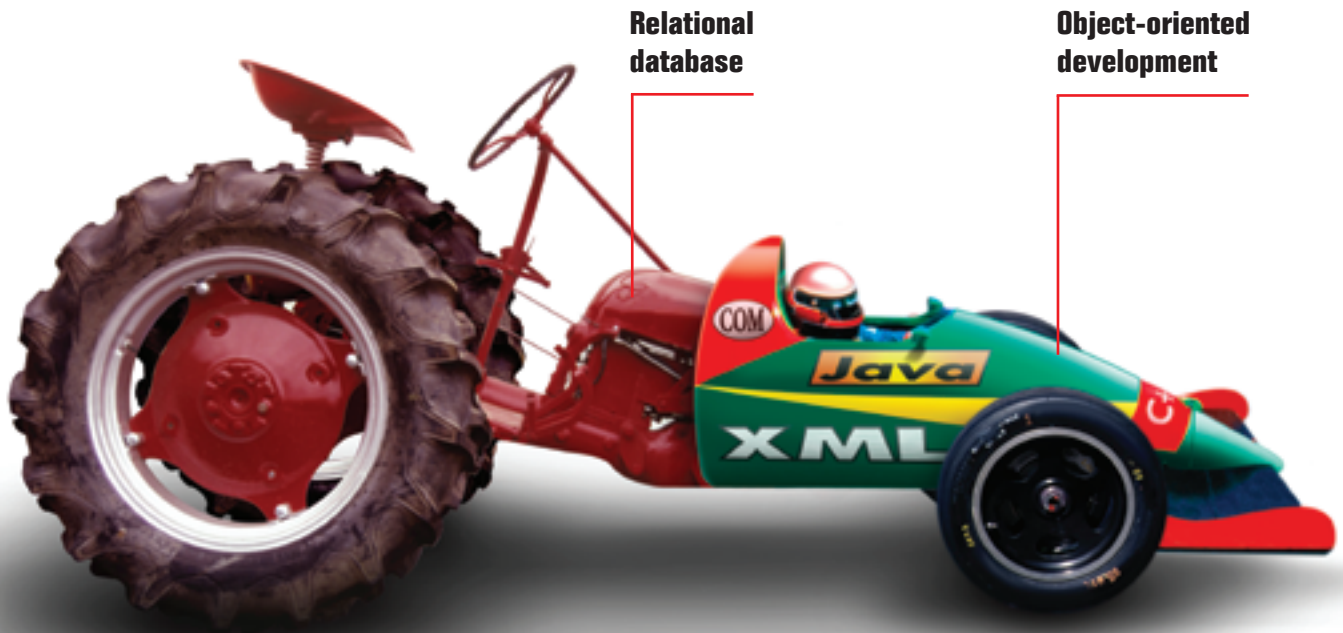


Figure 2 RRB class diagram



TIME TO CHANGE YOUR DATABASE

If your back-end database isn't a good match for your front-end development, you need a new database.

Caché, the *post-relational* database from InterSystems, combines high-performance SQL for faster queries and an advanced object database for rapidly storing and accessing objects. With Caché, no mapping is required between object and relational views of data. Every Caché class can be automatically projected as Java classes or EJB components with bean-managed persistence. Plus, every object class is instantly accessible as tables via ODBC and JDBC.

That means huge savings in both development and processing time. Applications built on Caché

are massively scalable and lightning fast. They require little or no database administration. And Caché's powerful Web application development environment dramatically reduces the time to build and modify applications.

We are InterSystems, a specialist in data management technology for over twenty-six years. We provide 24x7 support to four million users in 88 countries. Caché powers enterprise applications in healthcare, financial services, government, and many other sectors. Caché is available for Windows, OpenVMS, Linux, and major UNIX platforms – and it is deployed on systems ranging from two to over 10,000 simultaneous users.



Try a better database. For free.

Download a free, fully-functional, non-expiring version of Caché or request it on CD at www.InterSystems.com/match1

We finally got to the point where we can write a test application that uses RRB (see `EchoClientServer.java` and `EchoRRBClient.java`).

Our application got significantly simpler while having much more functionality. Now it has full timeout support. To test it, stop the echo server and you'll see your client's `onTimeout()` called for every timed-out request. Now let's see what the performance impact of the RRB is by running the same 1000 asynchronous requests/responses test:

```
java EchoRRBClient echo_server.properties 1000
...
N of reqs/resps = 1000
Total time = 5297 millisecs
Average roundtrip time = 5.297 millisecs
```

As you can see, there's no impact compared to our fastest non-RRB results.

RRB Proxies and RRB Chaining

So far we've discussed standalone JMS client applications that use a local RRB. It's a preferred mode, but RRB isn't limited to the local mode only. RRB is well suited to use in any distributed environment such as plain RMI applications or a J2EE container. That's where `RemoteCallable` and `JMSCallable` interfaces come into play. `RemoteCallable` is a remote interface, which means that the object that implements it isn't serialized when it's sent over the wire. Its remote reference is sent instead. `RemoteCallable` has the same methods as `LocallyCallable` but they can throw a `RemoteException` (see Listing 2).

You can call a remote RRB and supply a remote reference to the callback object exactly the same way that you'd do in a local RRB. There's a big caveat however. You're no longer guaranteed a callback! The explanation is simple. A remote RRB will try to call back your object through the remote reference, but if the network connection got lost it may not be able to do it. It's a known problem with a "server-based timeout implementation." The solution is to always have a local object that guarantees a callback. That's where `JMSCallable` comes into play.

Here's how it works. Client application always uses local RRB's `prepareJMSCallableRequest()` that creates a complete `RequestState` entry in the local RRB. It guarantees the callback. Besides that, `prepareJMSCallableRequest()` returns a wrapper object that implements a `JMSCallable` that can be sent over the wire (see `SampleProxyClientApp` and `ClientApp3.java` for the complete example). Note how `JMSCallable` is handled in RRB's implementation. Regardless of the number of RRBs in the request chain, the response will be sent directly to the first RRB in the chain (which is usually the client's local RRB) instead of traveling through the temporary queues of every RRB in chain.

The last thing I'd like to touch on is the use of RRB in what I call an "Asynchronous Service Facade (ACF)" implementation. Usually in an application with an asynchronous JMS server, you need to know and do quite a few things:

- You need to know the format of the message
- You need to know destination type (a queue or a topic) and its jndi name
- You need to create a connection, session, sender, receiver, and set a listener

RRB-enabled ACF provides a friendlier interface while preserving all the benefits of the asynchronous communication (see `SampleProxy.RrbProxy.MyProxyBean.ejb`, which is a stateless session bean facade for the asynchronous echo service). Here's how you can call it from the remote client application (see Listing 3). Listing 4 shows how to call a `MyProxyBean` blocking method. There's a caveat here. If you call RRB from a stateless session bean (SLSB) and block it until you get the response from the JMS, you have to set the transaction attribute for the SLSB to `NotSupported`, because if transactions are supported, JMS messages are sent at the transaction commit time.

Conclusion

Request Response Broker provides JMS application developers with a robust and easy-to-use implementation of the Request-Response paradigm. ☞

Listing 1 LocallyCallable interface

```
public interface LocallyCallable extends Callable
{
    // onResult callback
    public void onResult ( Object result, Object state );

    // onTimeout callback
    public void onTimeout ( Object state );
}
```

Listing 2 RemotelyCallable interface

```
public interface RemotelyCallable extends Callable, Remote
{
    // onResult callback
    public void onResult ( Object result, Object state ) throws
    RemoteException;

    // onTimeout callback
    public void onTimeout ( Object state ) throws RemoteException;
}
```

Listing 3 ClientApp3.java

```
// create client callback object
LocalClient clientcb = new LocalClient();
...
// call the remote method asynchronously
for ( int i=0; i<NUM_CALLS; i++ )
{
    // add request info to the local RRB and get JMSCallable
    wrapper back
    JMSCallable jmscb = rrb.prepareJMSCallableRequest ( cli-
    entcb,
                                                    1000,
    null );
    int res = proxy.doEcho( jmscb );
}
```

Listing 4 ClientApp4.java

```
// call the remote method synchronously
for ( int i=0; i<NUM_CALLS; i++ )
{
    Object res = proxy.doEchoBlocking ( "string" + i );
    System.out.println ( "result=" + res );
}
```




21 WAYS TO USE SPREADSHEETS IN YOUR JAVA APPLICATIONS

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

Formula One e.Spreadsheet 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!**

**21
WAYS TO USE
SPREADSHEETS
IN YOUR JAVA
APPLICATIONS**

ReportingEngines

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

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

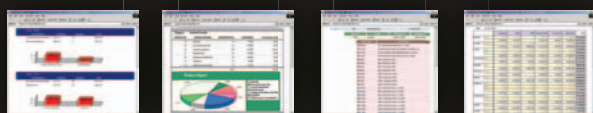


ReportingEngines

JAVA REPORTING TOOLS FROM ACTUATE

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>

Copyright © 2004 ReportingEngines (a division of Actuate Corporation). All rights reserved. Formula One is a registered trademark of Actuate Corporation. Java and Java-based trademarks and logos are the trademarks or registered trademarks of Sun Microsystems Inc., in the United States and other countries. All other trademarks are property of their respective owners. All specifications subject to change without notice.

Enterprise Content Management on the Java Platform

by Murali Kaundinya
and Sunil Mathew

A peek into Java standard APIs for accessing a content repository

Java Web applications have needed a standards-based API for Enterprise Content Management (ECM) for a long time. ECM is an essential requirement for Web applications on the Internet, intranets, and extranets. ECM vendors have proprietary APIs in various languages and this fact has inhibited ECM architectures from being interoperable. JSR-170 for ECM defines a new set of APIs to standardize the interface with ECM products. It aims to make the ECM product pluggable, much like the JDBC the API enables application code to be independent of databases products. JSR-170 has been actively supported by several ECM vendors and approved for public review. Its adoption is predicated on enterprises demanding it from ECM vendors, and it remains to be seen if these vendors will forego their unfair advantage.

In this article, we explain the lifecycle management services associated with "Content" to a Java developer building enterprise applications focusing on the new and emerging JSR-170. Enterprise Content Management (ECM) is about managing the lifecycle of "Content" in an enterprise. The lifecycle of managing such content requires a robust architecture. The lifecycle of "content," as shown in Figure 1, begins with its getting authored with some metadata. It's formally represented in digital form and uploaded to some server using Web protocols. It's then processed, which typically consists of sorting, classifying, and storing in a form that's subsequently easy to query and search. Content gets served to an authenticated and authorized user either in isolation or merged and aggregated with other content. Not all users are interested in the same kind of content so content has to be custom-

ized to suit individual user preferences, display device characteristics, and local and internationalization requirements. In Figure 1, users from various domains want to access content through various devices. The same content has to be rendered on various client devices. Compounding that, with the innumerable types of content and associated standards, a single general-purpose Content Management System (CMS) is seldom sufficient for an enterprise. Enterprise architectures deploy more than one ECM product, each having its own APIs for lifecycle management services, which increases developer complexity. A unified and standardized API such as JSR-170 can simplify the task of managing content across various vendor products and frameworks.

Enterprise Content Management

The ECM lifecycle comprises of a set of independent tasks, all of which are part of a large workflow. The workflow

begins by identifying the roles required for ECM lifecycle tasks and assigning users to those roles. Typical roles are content creators, reviewers, translators, classifiers, approvers, deployers, managers, etc. Once users, roles, and groups are identified, the data must be provisioned in the enterprise identity management repository. The next step involves content creation. Content creation gets done through tools that vary by content type. The creation exercise has localization and internationalization requirements. In an automated system, content can also be aggregated from multiple sources. Both the manual creation tasks and the automated tasks have to consider requirements for the transport protocol. Once the content gets submitted to a server, it has to get versioned. All content has to have metadata that describes the content's characteristics. It can be defined either at the time of creation by the content author, or it can be extracted at the



Murali Kaundinya is chief architect of Sun's Enterprise Web Services - Application Practice in the U.S. He helps customers with strategies on SOA and implements standards-based solutions.

murali.kaundinya@sun.com

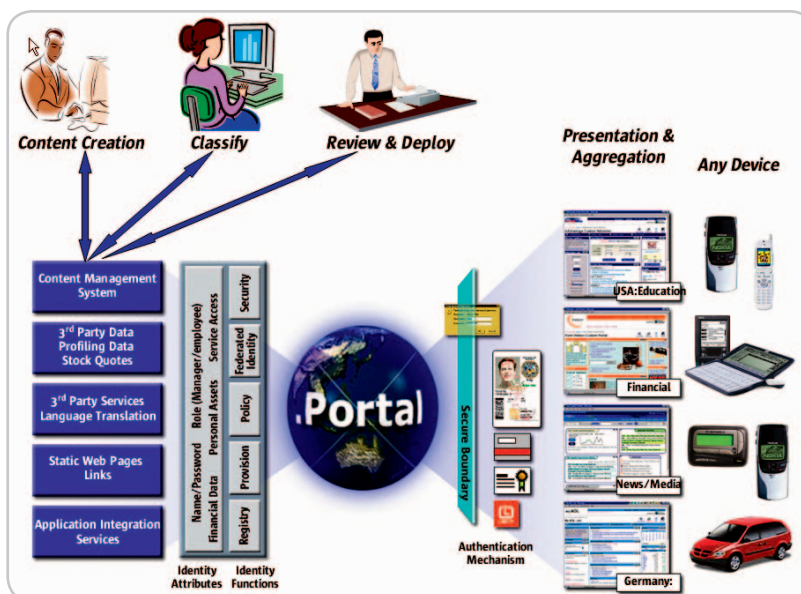


Figure 1 Lifecycle of "Content" in an enterprise

time of classification, or it can be both.

Once submitted content often has to be translated for an international audience. It may also have to be transformed based on visual formatting requirements specified in templates and other style sheets.

Once the content is transformed, it has to be assigned to the appropriate placeholders for dynamic rendering. The scope of ECM can also extend to content delivery. Delivery involves the assembly of dynamic content. It also requires the construction of an index and the ability to search the site for all of its content. There may be personalization requirements and consumers may have a preference about how they want the content to be structured. Consumers may have various authorization privileges based on their roles. Many of these content delivery requirements are also applicable to portal architectures. Portals use ECM solutions as a back-end service. The scope of this article is restricted to the Java interfaces dealing with content repositories.

Java Content Repository Model

The purpose of this API is to provide a standard implementation-independent way to access content bi-directionally on a granular level in a content repository. The challenge is to allow enough flexibility in the API so it can be used for hierarchical (path-based addressing) and non-hierarchical (UUID-based addressing) repository models. The APIs should

be easy to use from the programmer's point-of-view and at the same time its core focus should be to interface with a repository and not venture into areas that might be regarded as "content applications." ECM products have some common base features and they distinguish themselves with some unique features. One of the objectives of the API is to make it easy to implement on top of existing content repositories. The other objective is to standardize some of the more complex functionality needed by advanced content-related applications. To accelerate the adoption of this standard interface by ECM vendors, JSR-170 has taken a multi-step approach to the implementation of the APIs. Level 1 of the API defines a set of basic repository operations such as Read, Update, and Delete functions, the assigning of types to content items, serialization and search. Level 2 defines some of the advanced repository functions that are needed such as advanced content management like supporting transactions, versioning, access permissions, locking, and hard links between content items.

The repository as exposed through level 1 of the JCR is a tree structure very much like the Unix file system. It comprises nodes that can have zero or many child nodes. It should also support CRUD (Create/Read/Update/Delete) operations on the nodes and provide for assigning node types and the means to search the repository. Nodes can have zero

or more child properties. It should be possible to do retrieval and traversal of nodes and properties. A path-syntax has been defined to navigate the tree. The repository has three layers of isolation. javax.jcr.repository is an interface. An object implementing this interface represents a persistent data-store. javax.jcr.workspace is an interface; objects implementing this interface serve as a private view whose activities are only visible to users in this workspace. Changes made to this view have to be committed with an explicit checkin operation. A third type of isolation is between the workspace and the nodes (objects) in memory. A repository is similar to the well-known concurrent versioning system but there are some subtle differences. JCR doesn't distinguish, and rightfully so, between content and its metadata. It's up to the application to define its preferred conventions. JCR can be implemented on top of a file system, WebDAV, RDBMS, etc. Figure 2 shows a high-level JCR architecture.

An ECM application that's protected through JAAS retrieves a handle to a JCR Repository object using Java's Naming and Directory Interface. It populates a Credential object by pulling attributes from JAAS and invokes the Repository object's login method. So it retrieves a ticket that's like a session. Using the ticket, it retrieves one or more workspaces. The workspace provides for APIs to navigate the node tree and modify the nodes and their properties. JCR provides APIs to copy and move nodes around. It also lets APIs import and export nodes to external systems. A node can be serialized in an XML document. Likewise, an XML document compliant to some schema can be imported and attached to an existing tree. In a nutshell, JCR is similar to a Java DOM (Document Object Model) API with an ECM-friendly syntax.

As we said before, the motivation for having two levels of API for this JSR is so this complex set of APIs can be adopted by the industry in a phased way. A JCR repository is viewed as a collection of workspaces, each of which organizes the information in it in a graph (or tree) structure shown in Figure 3. Level 1 of the API defines a standard way to acquire a handle to a workspace in a



Sunil Mathew currently manages the Enterprise Web Services Practice for Sun Microsystems. Previously, Sunil has been a senior Java architect with Sun's Java Center group and he has led the Technology Solutions Architecture group for Sun in the northeast. He has over 15 years experience in Information Technology and is a coauthor of *Java Web Services Architecture*, Morgan Kaufmann Publishers.

sunil.mathew@sun.com

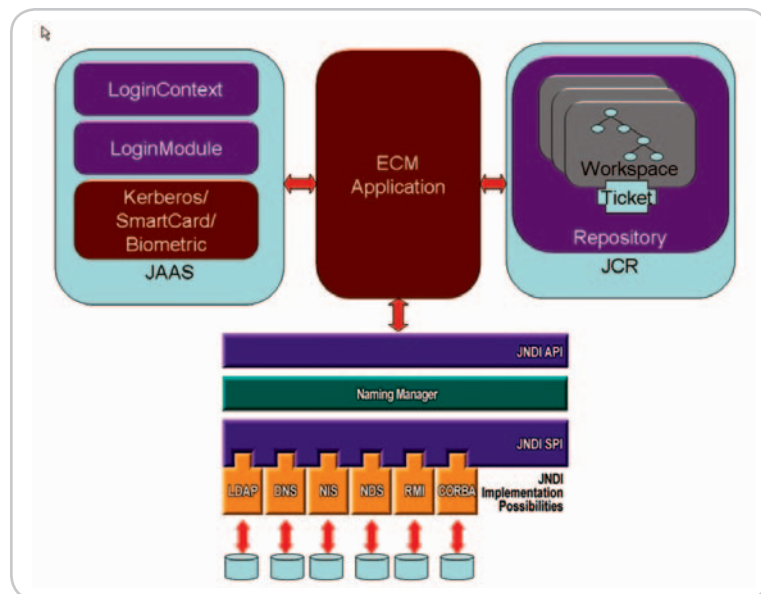


Figure 2 Architecture for Java Content Repository

“Java Web applications have needed a standards-based API for Enterprise Content Management (ECM) for a long time”

repository, to authenticate to the workspace, and to access or manipulate data in a workspace at the content-element level.

A client authenticates to a *Workspace* in a *Repository* by presenting a set of *Credentials*, and when authentication is successful gets a *Ticket*. Although it's not specified in the specification, a handle to a *Repository* is generally obtained via a JNDI lookup as shown in Figure 4. A *Ticket* can be thought of as a particular user's copy of a workspace. Every ticket must have a corresponding *Workspace* object associated with it but a workspace can have zero or more tickets associated with it. The information in a workspace (as represented as a tree or a graph) can be manipulated through a ticket as long as the user has permissions to make those manipulations. The user can also get a handle to

a workspace and directly manipulate the data in that workspace instead of using a ticket associated with the workspace. When a user manipulates data through the methods of a *Ticket* object, the data that's changed must be explicitly saved (through a *Ticket.save* call, for example) before it's persisted. Any data changes made directly to the *Workspace* object are automatically persisted.

The data in a workspace is organized as *Items* in a tree structure. Items are either *Nodes* or *Properties*. Properties can't have children and are leaves in the tree. The enterprise's content data is stored as values of properties. Nodes can have children that are other nodes or properties. Every node has one or more *node types* associated with it, one of which must be the *primary* node type associated with that node. A

primary node type defines certain characteristics (e.g., the types) that the node properties and child nodes are allowed and/or required to have, i.e., the node type enforces certain constraints on the node's children. Every node in the repository has a special property called *jcr:primaryType* that records the name of its primary node type. The primary node type uniformly enforces constraints on all child nodes and properties. If a child node is to have unique additional constraints or characteristics, the parent node may also be assigned one or more *mixin types*. Every node that has a *min* node type also has a system assigned property called *jcr:mixinTypes* that records its *mixin* node types.

As with XML document elements, name collisions for items in a repository is avoided by prefixing the item name with its namespace. The prefix is a short name for a URI (like in an XML document) and all JCR-compliant repositories have a namespace registry. There are served prefixes that are pre-assigned to certain namespaces. For example, the prefix *jcr* (e.g, *jcr:content*) is reserved for built-in node type and the prefix *mixin* is reserved for *mixin* node type. Figure 5 illustrates how a new namespace prefix can be registered before use. Reading data from a workspace consists of accessing the contained properties and extracting property values (the content). Properties can be accessed directly by specifying the path or by traversing the tree from a parent node. Level 1 API specification covers:

- Tree traversal to access nodes and properties
- Reading and writing property values
- Creation and deletion of items (nodes or properties)
- Identification of node types
- Searching the repository

Figure 6 illustrates a subset of the level 1 API for reading data from the repository. Each property has to have one of eight associated types. These types

```
Root
|-- header
|   |-- jcr: created = "2004-01-01"
|   |-- jcr: content
|       |-- acme: title = "ACME Corp."
|       |-- acme: line2= "Engineering consultants"
|-- footer
|   |--jcr: created = "2004-01-01"
|   |--jcr: content
|       |--acme:footnote1= "© copyright ACME corp. 1999"
```

The code examples in this article will refer to the above repository structure.

Figure 3 Information represented in graph (tree) structure

```
import java.rmi.*
import javax.jcr.*
...
//look up JNDI for a handle to the Repository object
Repository rep = (Repository) Naming.lookup ("JCR");

//Simple Credentials implements Credentials interface
Credentials credentials = new SimpleCredentials ("username", "password");

//Obtain a Ticket
try {
    Ticket workspaceTicket = rep.login (credentials, "myWorkspace");
} catch (LoginException e) {
    //possibly Bad credentials
} catch (NoSuchWorkspaceException) {
    //wrong workspace name given?
}
```

Figure 4 Accessing a handle to a repository through JNDI

if (wantToGoHomeSooner)



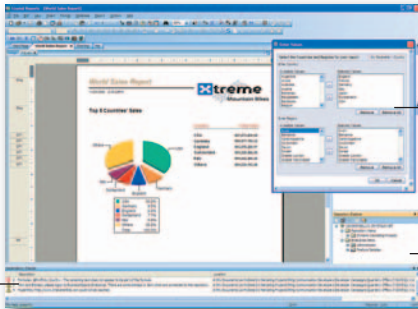
NEW Crystal Reports XI helps you spend less time integrating reports.

Visual Report Designer

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

NEW Dynamic Cascading Prompts

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



Report Distribution

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

Repository Explorer

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

NEW Report Dependency Checker

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

NEW Crystal Reports Server

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

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

NEW Free Runtime Licensing

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

Get home sooner with Crystal Reports XI.

Visit www.businessobjects.com/dev/p26

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

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





are represented by integer constants defined in the `PropertyType` class. They are `STRING`, `BINARY`, `DOUBLE`, `LONG`, `BOOLEAN`, `DATE`, `SOFTLINK`, and `REFERENCE`. A `SOFTLINK` is a special string type. It's a string representation of a repository path. A `REFERENCE` type is also a special String type. It's a node UUID. The UUID identifies a node in the workspace and, unlike a `SOFTLINK`, the UUID must be valid and must point to a particular node. It's important to note that not every node will have a UUID. Only nodes with *mix:referenceable* will have an associated unique identifier. This uid is stored as a value of the node property *jcr:uuid*. This property is read-only and is automatically assigned by the system when a node type of *mix:referenceable* is assigned to a node. Referenceable nodes aren't required for level 1 API compliance so their use is explained in detail in the next section.

The value of a property can be obtained, even if the client doesn't know the type of the property, by using the special holder class, *Value* (see Figure 7). Modifying the workspace takes the form of adding child nodes or properties to a node, removing a node or property, and modifying property values. When adding a node, the node type of that node can be specified. If the node type isn't specified, the system assigns the node type of the parent node. When setting a property, it's possible to specify the property name, property value, and property type. If the property type is omitted, the system tries to identify the property type from the Java type of the content. For example, if an input stream is provided, as in the example below, the system assumes a `BINARY` property type.

As mentioned above, changes made through the Ticket object aren't persisted until explicitly saved. If `Ticket.save()` is called after making changes

to one or more nodes and properties, all changes are persisted. If a `Node.save()` is invoked, all changes to that node and its children are persisted. If `node.save(true)` is invoked only changes to the node itself (not the changes to its children) are saved as shown in Figure 8. It's possible for a node to have multiple parents. To accommodate this, the Node interface API has a `Node.addExistingNode(<relpath>)` method to add an existing node as a child to a node. This method adds a child node that's a `REFERENCE` type. This means that the child node (the one with multiple parents) must have a UUID (i.e., a system property "*jcr:uuid*"). It should be noted that only nodes created with a mixin node type of *mix:referenceable* will have a system-generated uuid property, and so only these types of nodes can have multiple parents. When adding an existing node as a child to another node, a cyclical relationship (where a node becomes its own descendant) is prohibited; the system will throw a `ConstraintViolationException` if this is detected on the `addExistingNode()` call.

Level 2 of JCR requires that a provider provides several additional functionalities for compliance such as transactions, versioning, observation, access control, and locking. We'll provide greater details on these features in a future online version.

Summary

JCR is a refreshing and welcome addition to the realm of enterprise content management. The motivation is well-founded and the goals are well-defined and scoped. It has had active representation from several leading Content Management System vendors from conception all the way through the final vote. Some of the vendors, such as Jahia.org and Venetica.com, already offer compliant implementations. It can certainly make applications that interface with ECM a lot easier like JDBC did with databases. It remains to be seen how enterprises adopt this standard and demand compliance and support from leading vendors. OpenCMS, though unrelated to JSR-170, is another Open Source CMS standard available from openCMS.org.

```
//register "acme" as a valid namespace prefix in the workspace.
Workspace workspace = wrkspcTicket.getWorkspace();
Workspace.getNamespaceRegistry().setMapping("acme", "http://www.acme.com/content/1.0");
```

Figure 5 Namespace prefix being registered before use

```
//navigating the tree by first obtaining the root node
Node root = wrkspcTicket.getRootNode();
Node footer = root.getNode("footer");
Property p = footer.getProperty("acme: footnote1");

//print our string value of property ("copyright ACME corp. 1999")
System.out.println(p.getString());
//illustrates how the same property value can be accessed directly
p = root.getProperty("footer/acme: footnote1");
System.out.println(p.getString());
```

Figure 6 Illustration of reading data from repository

```
//navigating the tree by first obtaining the root node
Node root = wrkspcTicket.getRootNode();
Node footer = root.getNode("footer");
Property p = footer.getProperty("acme: footnote1");
Value v = p.getValue();
```

Figure 7 Accessing value without even knowing its type

```
//add a new node to the header node
Node root = wrkspcTicket.getRootNode();
Node header = root.getNode("header");
Node optionalHeader = Header.addNode("optional");
//assign a new binary property (input stream representing logo of Acme corp.)
Node contentNode = optionalHeader.addNode("jcr: content");
BufferedInputStream is = new BufferedInputStream(new FileInputStream("/home/logo.gif"));
contentNode.setProperty("logo", is);
//persist all changes
wrkspcTicket.save();
```

Figure 8 Persisting a node in isolation

Nothing beats our racks

Absolutely nothing



EV1Servers.net
an ev1.net company

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*** **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

Instant Activation!

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.

Using Java Data Mining to Develop Advanced Analytics Applications

by Sunil Venkayala

The predictive capabilities of enterprise Java apps

With the standardization of the Java Data Mining (JDM) API, Enterprise Java applications have been given predictive technologies.

Data mining is a widely accepted technology used for extracting hidden patterns from data. It is used to solve many business problems like identifying cross-sell or up-sell opportunities for specific customers based on customer profiles and purchase patterns, predicting which customers are likely to churn, creating effective product campaigns, detecting fraud, and finding natural segments.

More and more data mining algorithms are being embedded in databases. Advanced analytics, like data mining, is now widely integrated with applications. The objective of this article is to introduce Java developers to data mining and explain how the JDM standard can be used to integrate this technology with enterprise applications.



Sunil Venkayala is a J2EE and XML group leader and principal member of the technical staff in Oracle's Data Mining Technologies Group. He's an expert group member of the Java Data Mining (JDM) standard developed under JSR-73. Sunil has more than five years of experience in developing applications using predictive technologies available in the Oracle Database. He has more than seven years of experience working with Java and Internet technologies.

sunil.venkayala@gmail.com

Data Mining Functions

Data mining offers different techniques, *aka mining functions*, that can be used depending on the type of problem to be solved. For example, a marketing manager who wants to find out which customers are likely to buy a new product can use the classification function. Similarly a supermarket manager who wants to determine which products to put next to milk and eggs, or what coupons to issue to a given customer to promote the purchase of related items can use the association function.

Data mining functions are divided into two main types called *supervised* (directed) and *unsupervised* (undirected).

Supervised functions are used to predict a value. They require a user to specify a set of predictor attributes and a target attribute. Predictors are the attributes used to predict the target at-

tribute value. For example, a customer's age, address, occupation, and products purchased can be used to predict the target attribute "Will the customer buy the new product? (YES/NO)."

Classification and regression are categorized as supervised functions. Classification is used to predict discrete values, e.g., "buy" or "notBuy," and regression is used to predict continuous values, e.g., salary or price.

Unsupervised functions are used to find the intrinsic structure, relations, or affinities in data. Unsupervised mining doesn't use a target. Clustering and association functions come under this category. Clustering is used to find the natural groupings of data, and association is used to infer co-occurrence rules from the data.

The Data Mining Process

Typically data mining projects are initiated by a business problem. For example, a CEO could ask, "How can I target the right customers to maximize profits?" Once the business problem is defined, the next step is to understand the data available and select the appropriate data to solve the problem. Based on the data's characteristics, prepare the data for mining.

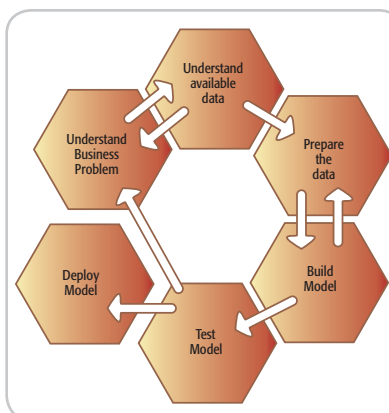


Figure 1 Data mining process

Select the right mining function and build a mining model with the data. After building the model, evaluate the model results. After evaluation, deploy the model. The CRISP-DM standard details the typical data mining process. Figure 1 illustrates a typical data mining process.

Enterprise applications like CRM analytics try to automate the data-mining process for common problems like intelligent marketing campaigns and market-basket analysis.

JDM API Overview

The Java Community Process (JCP) released the JDM 1.0 standard in August of 2004. JDM provides an industry standard API to integrate data mining functionality with applications. It facilitates the development of vendor-neutral data mining tools/solutions. It supports many commonly used mining functions and algorithms.

JDM uses the factory-method pattern to define Java interfaces that can be implemented in a vendor-neutral fashion. In the analytics business there's a broad set of data mining vendors who sell everything from a complete data mining solution to a single mining function. JDM conformance states that even a vendor with one algorithm/function can be JDM-conformant.

In JDM, *javax.datamining* is the base package that defines infrastructure interfaces and exception classes. Sub-packages are divided by mining function type, algorithm type, and core sub-packages. Core subpackages are *javax.datamining.resource*, *javax.datamining.base*, *javax.datamining.data*. The resource package defines connection-related interfaces that enable the applications to access Data Mining Engine (DME). The base package defines prime objects like mining model. The data package defines all physical

and logical data-related interfaces. The *javax.datamining.supervised* package defines the supervised function-related interfaces and the *javax.datamining.algorithm* package contains all mining algorithm subclass packages.

Solving the Customer Churn Problem Using JDM

Problem Definition

Customer attrition is one of the big problems companies face. Knowing which customers are likely to leave can be used to develop a customer-retention strategy. Using data-mining classifications one can predict which customers are likely to leave. In the telecommunications industry, this problem is known as customer churn. Churn is a measure of the number of customers who leave or switch to competitors.

Understand and Prepare the Data

Based on the problem and its scope, domain experts, data analysts, and database administrators (DBA) will be involved in understanding and preparing data for mining. Domain experts and data analysts specify the data required for solving the problem. A DBA collects it and provides it in the format the analyst asked for.

In the following example, several customer attributes are identified to solve the churn problem. For simplicity's sake, we'll look at 10 predictors. However, a real-world dataset could have hundreds or even thousands of attributes (see Table 1).

Here CUSTOMER_ID is used as the case id, which is the unique identifier of a customer. The CHURN column is the target, which is the attribute to be predicted. All other attributes are used as predictors. For each predictor, the attribute type needs to be defined based on data characteristics.

There are three types of attributes, i.e., categorical, numerical, and ordinal.

A categorical attribute is an attribute where the values correspond to discrete categories. For example, state is a categorical attribute with discrete values (CA, NY, MA, etc.). Categorical attributes are either non-ordered (nominal) like state and gender, or ordered (ordinal) such as high, medium, or low temperatures. A numerical attribute is an attribute whose values are numbers that are either integer or real. Numerical attribute values are continuous as opposed to discrete or categorical values.

For supervised problems like this, historical data must be split into two datasets, i.e., for building and testing the model. Model building and testing operations need historical data about both types of customers, i.e., those who already left and those who are loyal. The model-apply operation requires data about new customers, whose churn details are to be predicted.

The Data Mining Engine (DME)

In JDM, the Data Mining Engine (DME) is the server that provides infrastructure and offers a set of data mining services. Every vendor must have a DME. For example, a database vendor providing embedded data-mining functionality inside the database can refer to the database server as its data mining engine.

JDM provides a Connection object for connecting to the DME. Applications can use the JNDI service to register the DME connection factory to access a DME in a vendor-neutral approach. The *javax.datamining.resource.Connection* object is used to represent a DME connection, do data-mining operations in the DME, and get metadata from the DME.

Listing 1 illustrates how to connect to a DME using a connection factory that's registered in a JNDI server.

Describe the Data

In data mining, the ability to describe the physical and logical characteristics of the mining data for building a model is important.

JDM defines a detailed API for describing physical and logical characteristics of the mining data.

The *javax.datamining.data.PhysicalDataSet* object is used to encapsulate location details and physical characteristics of the data.

The *javax.datamining.data.LogicalData* object is used to encapsulate logical characteristics of the data.

A logical attribute can be defined for each physical attribute in the physical data set. A logical attribute defines the attribute type and the data preparation status. The data preparation status defines whether the data in a column is prepared or not prepared. Some vendors support internal preparation

of the data. In JDM, the physical data set and logical data are named objects, which can be stored in and retrieved from the DME.

Listing 2 illustrates how to create a *PhysicalDataSet* object and save it in the DME. Here *PhysicalDataSet* encapsulates "CHURN_BUILD_TABLE" details. In this table "CUSTOMER_ID" is used as the caseId.

Listing 3 illustrates how to create a *LogicalData* object and save it in the DME. Here *LogicalData* is used to specify the attribute types. Some vendors derive some of the logical characteristics of attributes from the physical data. So JDM specifies logical data as an optional feature that vendors can support. Logical data is an input for the model-build operation. Other operations like apply and test get this information from the model.

Build the Mining Model

One important function of data mining is the production of a model. A model can be supervised or unsupervised.

In JDM *javax.datamining.base.Model* is the base class for all model types. To produce a mining model, one of the key inputs is the build settings object.

javax.datamining.base.BuildSettings is the base class for all build-settings objects, it encapsulates the algorithm settings, function settings, and logical data. The JDM API defines the specialized build-settings classes for each mining function.

In this example, the *ClassificationSettings* object is used to build a classification model to classify churners.

Applications can select an algorithm that works best for solving a business problem. Selecting the best algorithm and its settings values requires some knowledge of how each

Attribute Name	Attribute Type	Data Type	Attribute Role
CUSTOMER_ID	N/A	INTEGER	Case Id
AGE	Numerical	INTEGER	Predictor
WORKCLASS	Categorical	String	Predictor
EDUCATION	Categorical	String	Predictor
MARITAL_STATUS	Categorical	String	Predictor
OCCUPATION	Categorical	String	Predictor
RACE	Categorical	String	Predictor
SEX	Categorical	String	Predictor
CAPITAL_GAIN	Numerical	Double	Predictor
CAPITAL_LOSS	Numerical	Double	Predictor
NATIVE_COUNTRY	Categorical	String	Predictor
CHURN	N/A	String	Target

Table 1 Churn analysis customer attributes

algorithm works and experimentation with different algorithms and settings. The JDM API defines the interfaces to represent the various mining algorithms.

In this example, we will use the decision-tree algorithm.

In JDM, *javax.datamining.algorithm.tree.TreeSettings* object is used for representing decision-tree algorithm setting. Some vendors support implicit algorithm selection based on function and data characteristics. In those cases, applications can build models without specifying the algorithm settings.

Listing 4 illustrates how to create classification settings and save them in the DME. Here a classification-settings object encapsulates the logical data, algorithm settings, and target attribute details to build the churn model. Here the decision-tree algorithm is used. For more details about the algorithm settings refer to the JDM API documentation.

Listing 5 illustrates how to build a mining model by executing the build task. Typically model building is a long-running operation, JDM defines a task object that encapsulates the input and output details of a mining operation. A task object can be executed asynchronously or synchronously by an application. Applications can monitor the task-execution status using an execution handle.

An execution-handle object is created when the task is submitted for execution. For more details about the task execution and the execution handle, refer to the JDM API documentation.

Here the build task is created by specifying the input physical dataset name, build settings name, and output model name. The build task is saved and executed asynchronously in the DME. Applications can either wait for the task to be completed, or execute the task and check the status later.

Test the Mining Model

After building a mining model, one can evaluate the model using different test methodologies. The JDM API defines industry standard testing methodologies for supervised models.

For a classification model like the churn model, the *ClassificationTestTask* is used to compute classification test metrics. This task encapsulates input model name, test data name, and metrics object name. It produces a *ClassificationTestMetrics* object that encapsulates the accuracy, confusion matrix, and lift metrics details that are computed using the model.

Accuracy provides an estimate of how accurately the model can predict the target. For example, 0.9 accuracy means the model can accurately predict the results 90% of the time.

Confusion matrix is a two-dimensional, $N \times N$ table that indicates the number of correct and incorrect predictions a classification model made on specific test data. It provides a measure of how well a classification model predicts the outcome and where it makes mistakes.

Lift is a measure of how much better prediction results are using a model as opposed to chance. To explain the lift we will use a product campaign example. Say product campaigning to all 100,000 existing customers results in sales of 10,000 products. However, by using the mining model say we sell 9,000 products by campaigning to only 30,000 selected customers. So by using the mining model, campaign efficiency is increased three times, so the lift value is computed as 3, i.e., $(9000/30000)/(10000/100000)$.

Listing 6 illustrates how to test the churn model by executing the classification test task using "CHURN_TEST_TABLE." After successfully completing the task, a classification test metrics object is created in the DME. It can be

retrieved from the DME to explore the test metrics. (Listings 6–8 can be downloaded from www.sys-con.com/java/sourcec.cfm.)

Apply the Mining Model

After evaluating the model, the model is ready to be deployed to make predictions. JDM provides an *ApplySettings* interface that encapsulates the settings related to the apply operation. The apply operation will result in an output table with the predictions for each case. Apply settings can be configured to produce different contents in the output table. For more details on apply settings, refer to JDM API documentation.

In this example, we use the top prediction apply setting to produce the top prediction for each case. The *DataSetApplyTask* is used to apply the churn model on the "CHURN_APPLY_TABLE." JDM supports *RecordApplyTask* to compute the prediction for a single record; this task is useful for real-time predictions. In this example, we use the dataset apply task to do the batch apply to make predictions for all the records in the "CHURN_APPLY_TABLE".

Listing 7 illustrates how to apply the "CHURN_MODEL" on "CHURN_APPLY_TABLE" to produce an output table "CHURN_APPLY_RESULTS" that will have the predicted churn value "YES or NO" for each customer.

After doing the apply task, a "CHURN_APPLY_RESULTS" table will be created with two columns, "CUSTOMER_ID" and "PREDICTED_CHURN." The probability associated with each prediction can be obtained by specifying it in the *ApplySettings*.

Here the *mapTopPrediction* method is used to map the top prediction value to the column name. The source destination map is used to carry over some of the columns from the input table to the apply-output table along with the prediction columns. In this case, "CUSTOMER_ID" column is carried over from the apply-input table to the output table. JDM specifies many other output formats so applications can generate the apply-output table in the required format. A discussion of all the available options is beyond the scope of this article.

Figure 2 summarizes the JDM data mining process flow that we did in this example.

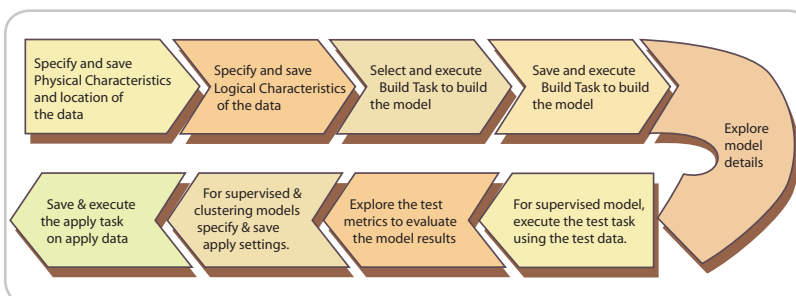


Figure 2 JDM data mining process flow

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.



Market Basket Analysis Example

To explain the use of unsupervised data mining in a practical scenario, we'll use one of the most popular data mining problems called *market basket analysis*.

The purpose of market basket analysis is to determine what products customers buy together. Knowing what products people buy together can be helpful to traditional retailers and web stores like Amazon.

The information can be used to design store layouts, web page designs, and catalog designs by keeping all cross-sell and up-sell products together. It can also be used in product promotions like discounts for cross-sell or up-sell products. Direct marketers can use basket analysis results to decide what new products to offer their prior customers.

To do market basket analysis, it's necessary to list the transactions customers made. Sometimes customer demographics and promotion/discount details are used to infer rules related to demographics and promotions. Here we use five transactions at a pizza store. For simplicity's sake, we'll ignore the demographics and promotion/discount details.

Transaction 1: Pepperoni Pizza, Diet Coke

Transaction 2: Buffalo wings, Diet Coke

Transaction_Id	Product_name
1	Pepperoni Pizza
1	Diet Coke
1	Buffalo Wings
2	Buffalo Wings
2	Diet Coke
3	Pepperoni Pizza
3	Diet Coke
4	Diet Coke
4	French Fries
5	Diet Coke
5	Buffalo Wings

Table 2 PRODUCT_TRANSACTIONS

Rule ID	If (condition)	Then (association)	Confidence	Support
1	Buffalo Wings	Diet Coke	1	0.6
2	Pepperoni Pizza	Diet Coke	1	0.4
3	Buffalo Wings and Pepperoni Pizza	Diet Coke	1	0.2
4	French Fries	Diet Coke	1	0.2
5	Diet Coke	Buffalo Wings	0.6	0.6

Table 3 Association rules

Transaction 3: Pepperoni Pizza, Diet Coke

Transaction 4: Diet Coke, French Fries

Transaction 5: Diet Coke, Buffalo wings

The first step is to transform the transaction data above into a transactional format, i.e., a table with transaction id and product name columns. The table will look like Table 2. Only the items purchased are listed.

An association function is used for market basket analysis. An association model extracts the rules stating the support and confidence in each rule. The user can specify the minimum support, minimum confidence, and maximum rule length as build settings before building the model.

Since we have only five transactions, we'll build a model to extract all the possible rules by specifying minimum support as 0.1, minimum confidence as 0.51, and no maximum limit for the rule length. This model produces five rules (see Table 3).

In a typical scenario, you may have millions of transactions with thousands of products, so understanding the support and confidence measures and how these are calculated provides good insight into which rules need to be selected for a business problem.

Support is the percentage of records containing the item combination compared to the total number of records. For example take Rule 1, which says, "If Buffalo wings are purchased then diet coke will also be purchased." To calculate the support for this rule, we need to know how many of the five transactions conform to the rule. Actually, three transactions, i.e., 1, 2 and 5, conform to it. So the support for this rule is $3/5=0.6$.

Confidence of an association rule is the support for the combination divided by the support for the condition. Support gives an incomplete measure of the quality of an association rule. If you compare Rule 1 with Rule 5, both of them have the same support, i.e., 0.6, because support is not directional. Confidence is directional,

so that makes Rule 1 a better rule than Rule 5.

Rule length can be used to limit the length of the rules. When there are thousands of items/products with millions of transactions, rules get complex and lengthy, so it's used to limit the length of the rules in a model.

Using JDM to Solve the Market Basket Problem

So how does one use JDM API to build an association rules model and extract the appropriate rules from the model?

Typically data for association rules will be in a transactional format. A transactional format table will have three columns: "case id," "attribute name," and "attribute value" columns.

In JDM by using the *PhysicalAttributeRole* enumeration, the transactional format data can be described. The *AssociationSettings* interface is used to specify the build settings for association rules. It has minimum support, minimum confidence, and maximum rule-length settings that can be used to control the size of association rules model.

Listing 8 illustrates building a market-basket analysis model using the JDM association function and exploring the rules from the model using rule filters.

Conclusion

The use of data mining to solve business problems is on the upswing. JDM provides a standard Java interface for developing vendor-neutral data-mining applications. JDM supports common data-mining operations, as well as the creation, persistence, access, and maintenance of the metadata supporting mining activities. Oracle initiated a new JSR-247 to work on new features for a future version of the JDM standard. ☛

References

- *Java Data Mining Specification*. <http://jcp.org/aboutjava/communityprocess/final/jsr073/index.html>
- *Java Data Mining API Javadoc*. <http://www.oracle.com/technology/products/bi/odm/JSR-73/index.html>
- *Java Data Mining Project Home*. <https://datamining.dev.java.net>
- *Cross-Industry Standard Process for Data Mining (CRISP-DM)*. <http://www.crisp-dm.org>
- *JSR-247*. <http://jcp.org/en/jsr/detail?id=247>

Listing 1

```
//Get Connection factory object from the JNDI server
javax.datamining.resource.ConnectionFactory jdmCFactory =
(ConnectionFactory) jndiContext.lookup("java:comp/env/jdm/
MyServer");

//Create a data mining server connection specification
ConnectionSpec svrConnSpec = (javax.datamining.resource.
ConnectionSpec)

    jdmCFactory.getConnectionSpec();
svrConnSpec.setName( luseri );
svrConnSpec.setPassword( lpasswordi );
svrConnSpec.setURI( lserverURIi );

//Create a Data Mining Engine (DME) Connection
javax.datamining.resource.Connection dmeConn =
(javax.datamining.resource.Connection)jdmCFactory.getConnection(
    svrConnSpec );
```

Listing 2

```
//Get PhysicalDataSetFactory
PhysicalDataSetFactory pdsFactory =
    (PhysicalDataSetFactory)dmeConn.getFactory(
        ljavax.datamining.data.PhysicalDataSeti);

//Create PhysicalDataSet by settings import flag set to true
//to import the physical attributes details derived from
//the meta-data of the table.
PhysicalDataSet pdsBuild = pdsFactory.create(
    lDMUSER.CHURN_BUILD_TABLEi, true);
//Set the role of the CUSTOMER_ID column as caseId
PhysicalAttribute caseId = pdsBuild.getAttribute(lCUSTOMER_IDi);
//Save the PhysicalDataSet in the DME
dmeConn.saveObject(pdsBuild, lCHURN_BUILD_PDSi);
```

Listing 3

```
//Get LogicalDataFactory
LogicalDataFactory ldFactory =
    (LogicalDataFactory)dmeConn.getFactory(
        ljavax.datamining.data.LogicalData1);
//Create LogicalData by specifying BUILD_PDS as input
LogicalData ldBuild = ldFactory.create(lBUILD_PDSi);
//Set the attribute types of all the logical attributes
LogicalAttribute ageAttr = ldBuild.getAttribute(lAGEi);
ageAttr.setAttributeType(AttributeType.numerical);
LogicalAttribute workClassAttr = ldBuild.getAttribute(lWORKCLASSi);
ageAttr.setAttributeType(AttributeType.categorical);

. . . .
```

```
//Save LogicalData in the DME
dmeConn.saveObject(ldBuild, lCHURN_LD1);
```

Listing 4

```
//Create a classification settings factory
ClassificationSettingsFactory clasFactory =
    (ClassificationSettingsFactory)dmeConn.getFactory(
        "javax.datamining.supervised.classification.ClassificationSett
ings");
//Create a ClassificationSettings object
ClassificationSettings clas = clasFactory.create();
//Set Logical data
clas.setLogicalDataName(lCHURN_LD1);
//Set target attribute name
clas.setTargetAttributeName(lCHURN1);
//Create a TreeSettingsFactory
TreeSettingsFactory treeFactory =
    (TreeSettingsFactory)dmeConn.getFactory(
        "javax.datamining.algorithm.tree.TreeSettings");
//Create TreeSettings instance
TreeSettings treeAlgo = treeFactory.create();
treeAlgo.setBuildHomogeneityMetric(TreeHomogeneityMetric.entropy);
treeAlgo.setMaxDepth(10);
treeAlgo.setMinNodeSize( 10, SizeUnit.count );
//Set algorithm settings in the classification settings
clas.setAlgorithmSettings(treeAlgo);
//Save the build settings object in the database
dmeConn.saveObject(lCHURN_BUILD_SETTINGSi, clas, true);
```

Listing 5

```
//Create BuildTaskFactory
BuildTaskFactory buildTaskFactory = BuildTaskFactory)dmeConn.get-
Factory(
    ljavax.datamining.task.BuildTaski);
//Create BuildTask object
BuildTask buildTask = buildTaskFactory.create(
    lCHURN_BUILD_PDSi, lCHURN_BUILD_SETTINGSi, lCHURN_MODELi);
//Save BuildTask object
dmeConn.saveObject(lCHURN_BUILD_TASKi, buildTask, true);
//Execute build task asynchronously in the database
ExecutionHandle execHandle = dmeConn.execute(lCHURN_BUILD_TASKi);
//Wait for completion of the task
ExecutionStatus execStatus =
    execHandle.waitForCompletion(Integer.MAX_VALUE);
```




Calvin Austin

Core and Internals Editor

Java: What Does the Future Hold?

Last month I took a trip down memory lane, revisiting the history of J2SE. Apart from trying to remember key events, squeezing 10 years of history into one page was a challenge. I had to relegate many significant technologies to a sentence or two and some I didn't cover at all. However, looking at the future of Java is like looking at NASA's Apollo plans after the first walk on the moon. Yes, there will be valuable missions or releases, but even NASA canceled the last three moon missions due to budget issues.

The good news is that mission control has already have scoped out the 6.0 and 7.0 releases that will extend the Java release roadmap to 2008. Beyond the planned features in 6.0 that I am covering here, Sun has already hinted at Java's future through some recent statements and comments – a renewed focus on scripting languages, and new, but still not open source, Java licensing. Although the rework of the licenses is a good step, the opportunity to make a real statement has been missed. Both the JVM specification and Java language specifications are now mature enough to be handed to an external standards body, something that was promised in the early days of Java and I feel that now would be a good time to make whole on that commitment to developers worldwide.

Back to J2SE 6.0, code named Mustang. Planning for this release started in the summer of 2004 and is due to be shipped in the middle of 2006. JDK releases normally stick to the end of a quarter if possible, so that would mean a release date of end of June 2006.

The proposed JSR list included with the umbrella Java Specification Request 270 contains many familiar specifications. Why? Because a good portion of them were also slated for the Tiger release but were dropped for one reason or another.

Before I mention the JSRs that are included, it's worth mentioning one that is not there – JSR 203, the enhancements to New IO. The Google EC member mentioned that JSR 203, which was slated for Tiger, has now missed 6.0, which could mean it would be 2008 before this JSR is ever released. Realistically, those users who needed functionality like asynchronous IO have probably made alternate plans by now; waiting four more years is too long for even the most patient developer.

J2SE 6.0 and Web Services

The lion's share of the functional updates to J2SE 6.0 are in Web services. The plan is that J2SE will be the delivery platform for JSR 105, the XML digital signature API; JSR 222, the XML data binding JAXB 2.0 API; JSR 224, the JAX-RPC 2.0 API; and a penciled update to the core XML platform. By default J2EE will also inherit these specifications in its next release.

J2SE 6.0 and the Java Compiler

J2SE 5.0 contained many changes to the language and compiler so it wasn't too surprising that two JSRs that missed the J2SE 5.0 train are now slated for J2SE 6.0. The first was an interface to the Java compiler, JSR 199, and the second was a part of JSR 202 that included changes to the Java verifier. The compiler API was originally thought to be necessary for JSP pages and Ant, however, those technologies now use a well-known interface into the compiler. The updated specification contains an API that exports dependency information that would be useful with many build tools. The split verifier as described in JSR 202 is already used in J2ME land; it improves verification by generating verification tables at compile time and then using this information at runtime. The runtime

verification is therefore faster, however, the tradeoff is a larger class file. How this impacts large applications is still to be seen as is the question about whether the JDK should verify its own class files.

One new JSR for J2SE 6.0 is JSR 269, the Annotation Processing API. This builds on the addition of annotations into J2SE 5.0. Its use is primarily for tools, but developers will get to see the high-level benefits of it.

Is This Your Card?

I guess I'm as surprised as some others that a smart card API is really needed in the J2SE platform as proposed in JSR 268. Sun employee badges have embedded smart cards but the leap toward making every JVM in the world require additional code seems grandiose. My own desktop at Sun didn't support smart cards so testing this even at Sun is going to cause many headaches.

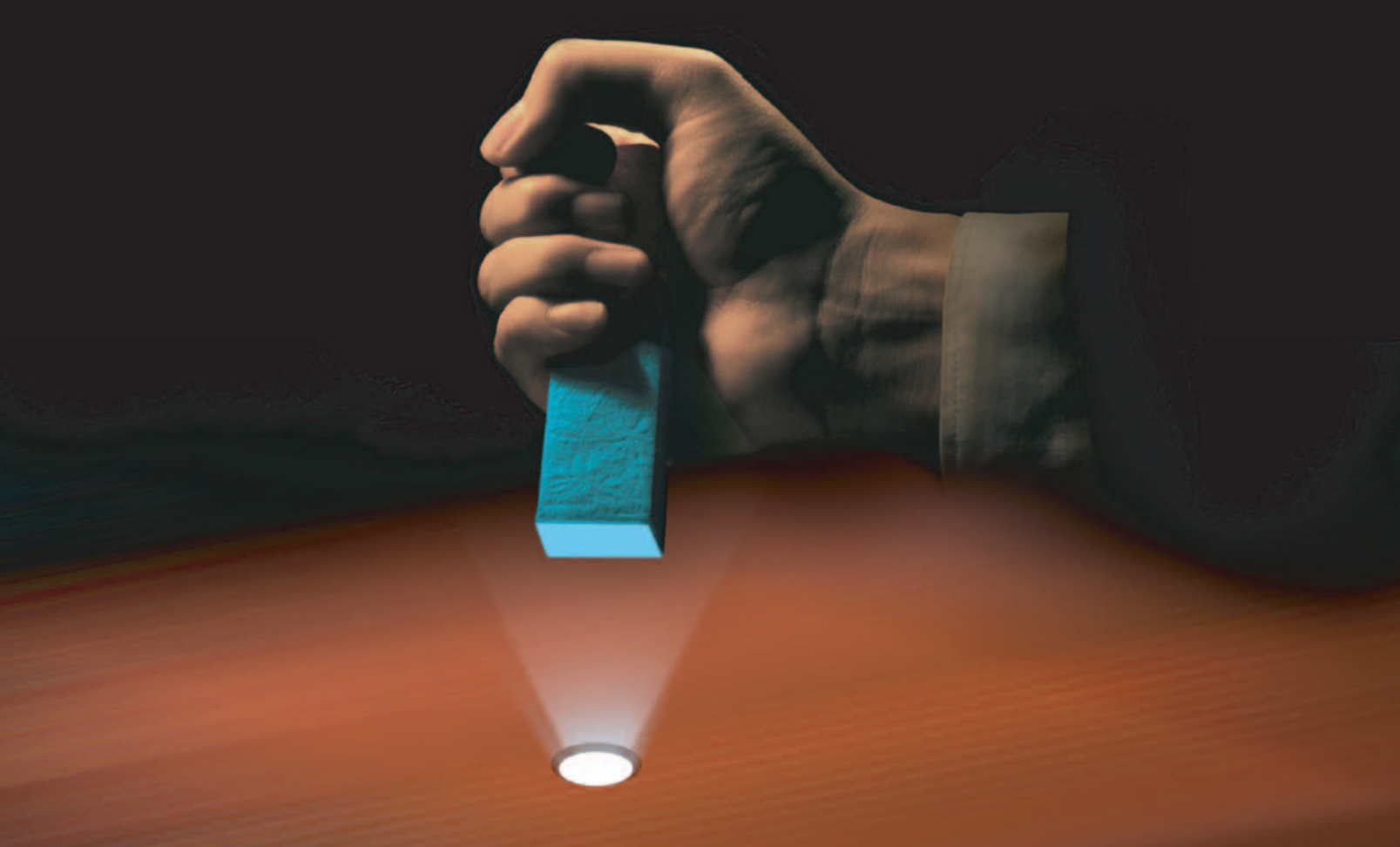
JSR 223, Scripting for the Java platform, is designed to expose Java objects to scripting languages like PHP. PHP 5.0 is already object oriented; whether those users will make the effort to learn Java is anyone's guess.

Wrapping Up

The last two proposed JSRs are JDBC 4.0 (JSR 221) and Javadoc changes (JSR 260). JDBC 4.0 builds on J2SE 5.0 metadata and generics to improve data access and management. JSR 260, Javadoc tag improvements, is primarily driven to improve the documentation of J2SE and the tools that generate Javadoc.

In closing, the future of J2SE, like those Apollo missions after the first moon landing, may not make history but will no doubt be of value. However, those trips to the moon will eventually move to the back burner so it's doubtful that any future improvements will be like rocket science. ☺

A section editor of JDI 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



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

Think JReport.

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

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

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

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

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

When it comes to embedded Java reporting, JReport is the perfect fit. Download a FREE copy of JReport today at www.jinfonet.com/jp4.

 **JReport**
JINFONET SOFTWARE

Taming Team Conflicts in Java Development

– Control your code –

by Brian Duff

A vital aspect of developing software as a team is the process of managing change effectively. Today, many teams leverage software tools to reduce the manual burden of coping with fluidity in software development. A version control system such as the Concurrent Versioning System (CVS) is an important tool in any team's arsenal of utilities and processes for change management.

Version control is useful, if not vital, to the way most teams cope with change. Over time, version control products have become increasingly sophisticated and flexible. A watershed event in the history of version control software was the advent of the CVS and its support for working in parallel.

Older version control systems employed a reservation model that let only one developer work on a file at a time. In a version control system that supports working in parallel, this restriction is relaxed. Working in parallel isn't just likely but certain in most teams. However, there's a trade-off. Besides managing change itself, your tool and processes must be capable of managing conflicts.

How Conflicts Happen

This article looks at version control conflicts from the point-of-view of a development team working with CVS and Java. Although many of the practices described here are applicable to most languages and version control tools, it's useful to include real examples in considering conflict-taming techniques.

To start thinking about how to reduce potential conflicts in a team, it's worthwhile to know a bit about how CVS and other version control systems merge parallel changes.

Imagine a situation in which you've modified an Ant build file locally and want to pull any changes other developers have made into your local copy. To do this, you issue the command `cvs update build.xml`. At this point, there are three possibilities:

- **No newer revision of the file exists in the repository.** There's nothing to do here. `cvs update` prints M for the file just to remind you that the local copy is modified.
- **A newer revision of the file exists, and all changes can be merged automatically.** The local copy of the file is replaced

with the merged version, and `cvs update` prints M for the file. Normally, a backup of the original copy is made and `cvs update` tells you its filename. Usually you don't need to do anything, but it's worth inspecting the merged file to make sure it's still syntactically and semantically correct.

- **A newer revision of the file exists, but the changes you've made locally conflict.** `cvs update` prints C for these files. Any non-conflicting changes are merged automatically. CVS marks conflicts in the file, and you have to resolve them.

Under which circumstances can changes be merged automatically, and how are conflicts detected? Most version control systems, including CVS, use a process called three-way merge. In merging, there are three relevant versions of a given file as shown in Figure 1. The *source* of a merge is the newest revision of the file. It includes all the changes made by other developers since you last updated the file. The *target* of a merge is the modified file you have in your CVS sandbox. The *base* or *common ancestor* revision is the revision your local sandbox changes were based on. In other words, it's the most recent revision you updated to or checked out.

The three-way merge process compares each of these contributors. If a block of text is different in all three revisions, there's a conflict. Otherwise, the changes can be merged automatically. For simplicity's sake, we'll just consider a block a single line, although merge algorithms are typically sophisticated enough to identify distinct regions of change that are several lines long.

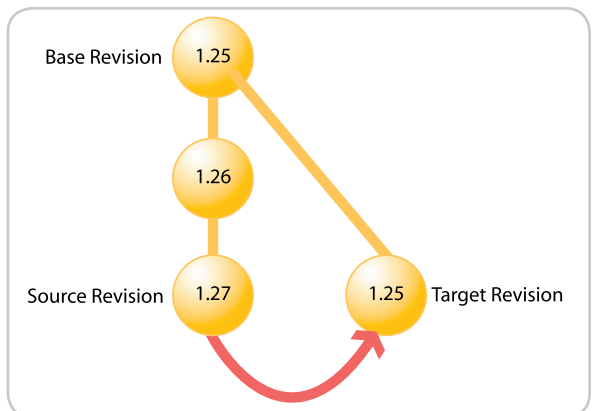


Figure 1



Brian Duff is a principal member of Oracle's JDeveloper technical staff. You can read his blog at <http://www.orablogs.com/duffblog>.

brian.duff@oracle.com

Table 1 provides an example using an Ant build file.

Here the project name has been changed locally from “jwidget” to “jsprocket” in line 1. This change can be automatically resolved, because it’s different in only one contributor. On the other hand, the value of the “main” property was changed locally to “org.acme.JSprocket” on line 3, but this conflicts with a change by another developer in revision 1.27. Line 3 is different in all three contributions.

When a conflict happens, a merge tool doesn’t have enough information to decide whether the changes you made locally or the changes made by the other developers should be left in the file. It requires good judgment and communication between team members to determine how to handle a conflict.

Dealing with Conflicts

Armed with knowledge of how conflicts occur, the next step is to consider whether there are general principles we can apply to avoid conflicts.

The most obvious solution is to avoid parallel changes altogether by enforcing a reserved model in which only a single developer is allowed to make changes to a file. At first this may seem like an ideal solution to the problem, but it’s not really a solution at all. Parallel changes are inevitable, particularly in a highly connected language such as Java.

Consider a Java developer who’s changing an interface that may be implemented by many classes in a complex system. To ensure that his changes won’t break the code on the main branch, he must make changes in every class that implements the interface. Meantime, another developer may want to add a method to one of those classes for a different task he’s working on. In a reserved model, one of these developers would be blocked, waiting for the other developers to finish.

A better solution is to allow parallel change but try to minimize the risk that conflicts will occur.

Organize Imports

Import statements change a lot. One developer may add a method or two in a Java class that would otherwise not conflict with a change made by a different developer, but there’s a good chance that those two developers may add import statements to the same part of the Java file. There are two ways of reducing the frequency of import statement conflicts.

The first approach is to use wide imports. This follows a general principle of minimizing the number of possible conflict zones in a file. If a developer introduces usage of a class that happens to be in a package for which there is already a wide import, there’s no need to update the imports. Hence any likelihood of a conflict is eliminated.

However, many Java developers prefer to use narrow imports, because they provide a more accurate description of the dependencies of a particular class on classes outside its package. If narrow imports are organized fairly haphazardly in the file or if new imports are always added to the end of the existing imports, conflicts will occur frequently. A good approach is to sort imports alphabetically and always insert new imports in their correct position in relation to existing imports.

To see how this works, consider two developers who have simultaneously introduced a new import to an existing class (see Table 2).

Because both import statements were added at the end of the existing set of statements, a conflict occurs. However, if the imports are sorted and are not alphabetically adjacent, no conflict will occur (see Table 3).

Typically, teams have some kind of coding conventions or style guide. A common convention limits the length of code lines to some specific fixed number of characters (80 is typical because of the traditional character width of terminals).

I’ve often heard arguments that the 80-character line length limit is archaic and obsolete. Some other programmers go through all kinds of contortions just so that no line exceeds the 80-character limit by even one character. The 80-character limit is good for readability, because it represents a lowest common denominator. With the possible exception of

most cell phone displays, almost every modern display is capable of comfortably showing at least 80 characters.

A line length limit reduces potential conflicts. As a general rule, the less information there is on a single line, the less likely a conflict will occur. This works because most diff and merge algorithms consider text files a sequence of lines.

Take two developers who are concurrently changing the implementation of a method. One decides to rename the method, and the other adjusts one of its parameters (see Table 4).

The two developers have made completely concurrent changes that should work fine, but these changes will result in a conflict. If the developers had reduced the amount of information on each line, an automatic merge would have been possible (see Table 5).

This is a rather contrived example, and it’s best to use good judgment. There is a trade-off between decreasing the potential for merge conflicts and maintaining code readability.

Do Not Version-Control 100% Generated Code

My team has a useful utility we sometimes use to generate user interface code. We describe the layout of a panel in XML, which is transformed to pretty gnarly GridBagLayout code when our product is built. Similarly, we generate subclasses of ArrayResourceBundle based on a properties file to make translatable resource lookups more efficient.

The important thing these types of files have in common is that they can be completely generated. From time to time, though, someone checks one of the generated output files into version control, and then the fun and games begin.

The first problem with this is the obvious one of duality. These generators have an input file and a result. The input file is a source file, and the output is something derived from the source. In version control parlance, the output is a derived object. Every time you change (create a new version of) the source file, you must also create a new version of the derived object. If they get out of sync, someone in your team is going to get very confused.

Revision 1.25 (Base)	Revision 1.27 (Source)	Local (Target)
1 <project name="jsprocket">	1 <project name="jwidget">	1 <project name="jwidget">
2 <property name="main">	2 <property > name="main"	2 <property name="main">
3 org.acme.JWidget	3 com.acme.JWidget	3 org.acme.JSprocket
4 </property>	4 </property>	4 </property>
5 </project>	5 </project>	5 </project>

Table 1

Developer A	Developer B
import java.io.IOException;	import javax.swing.JFrame;
import java.io.InputStream;	import java.io.IOException;
import javax.swing.JFrame;	import javax.swing.JPanel;

Table 2

Developer A	Developer B
import java.io.InputStream;	import java.io.IOException;
import javax.swing.JFrame;	import java.io.IOException;
import javax.swing.JFrame;	import javax.swing.JPanel;

Table 3

Original	Developer A	Developer B
paintCar(Color.RED);	paintCar(Color.GREEN);	car.paint(Color.RED);

Table 4

Original	Developer A	Developer B
paintCar(Color.RED);	paintCar(Color.GREEN);	car.paint(Color.RED);

Table 5



In general, derived objects shouldn't be version-controlled like other files. For the same reason, .class and .jar files aren't typically stored in a version control system unless they constitute a fixed dependency your project has on some other software.

Utilities that generate code often generate a lot of it in relation to the size of the original source file. This is part of the reason why code generators are used in the first place, but it means that a tiny change in a source file can cause a large ripple of changes throughout a generated file. When two developers do this in parallel, their small, easily merged changes in the source code may become a large set of difficult merge conflicts in the derived objects.

Avoid Magic in XML Grammar

Many modern Java applications make considerable use of XML.

Often the grammar of XML files you work with in Java is already well defined, but there's a great deal of flexibility in layout and formatting of XML documents, so you can apply some of the general suggestions in this article to XML too. For example, keeping lines short in XML may mean splitting up elements so that each attribute starts on a separate line. Sorting XML elements in a consistent order if the grammar allows it is also a technique that helps avoid conflicts.

If you are designing your own XML grammar for a Java application or for a tool you use to build your application, there's another important guideline about how to keep XML files both mergeable and readable. Consider whether the users of an application or tool will be version-controlling the XML files. If there's a chance users will, then a tool should strive to write XML in a way that limits conflicts.

A common problem with custom XML grammars is when a technique sometimes called "magic encoding" is used. Encoding, in this instance, doesn't pertain specifically to the character set encoding of an XML document but to the way data is stored in a particular instance document.

XML is a way of structuring data in a standard way that can be ubiquitously parsed. It's almost always better to use the structure inherent in XML itself to encode information you want to store or transfer in an XML file, rather than using another encoding inside an attribute or a text node.

Imagine an application that creates an XML file representing a diagram. Elements in this file define all shapes visible on the surface of the diagram. Each shape contains information about its position on the diagram, dimensions, color, and font. You may have internal data structures or parsing routines in your code that make it convenient to generate XML that looks something like this:

```
1 <diagram>
2 <shape id="0001">
3   <location>10, 10</location>
4   <size>50, 50</size>
5   <color>65280</color>
6   <font>Tahoma|BOLD|8</font>
7 </shape>
8 </diagram>
```

Each property of the shape contains structured information that is encoded in some format other than XML. Many of these

values will nevertheless make sense to someone editing your file, but color is a "magic" value that probably makes sense only to your color-parsing code and to programmers (who might need to do a bit of binary arithmetic to figure it out). There are two main problems with custom encoding structured information inside an XML file shared in a team:

- **Depending on the custom encoding you use, conflicts can increase.** In the previous example, if I move the shape up and you move it left, we have a conflict. If you decrease the font size and I change the typeface, we have a conflict.
- **Because your file is shared in a team, you must never assume that your file will only be edited by a tool.** If a conflict happens, users will have to edit the file manually. Several parsers of encoded structured information are used to process the preceding file. In other words, there are multiple points of failure. Is your location parser robust enough to cope with two commas? What about white space? Magic values are most problematic of all for user edits. Given a choice in a conflict between the colors 65280 and 272, which do you choose?

We could rewrite our XML to use XML syntax itself to encode the structured information.

```
1 <diagram>
2   <shape id="0001">
3     <location x="10"
4       y="10" />
5     <size width="50"
6       height="50" />
7     <color red="FF"
8       green="02"
9       blue="23"
10      alpha="00" />
11    <font face="Tahoma"
12      bold="true"
13      size="8" />
14  </shape>
15 </diagram>
```

This makes our diagram file far easier to merge if someone using our tool decides to start version-controlling diagram files.

Conclusion

This article highlighted a few techniques for making Java development and tools work better in a team environment. Many topics, such as merge tools that understand the specific syntax of Java or XML code to make conflict resolution more intelligent, weren't covered here. But with a few simple rules of thumb and the standard tools shipped with your version control software, conflicts in Java team development can be reduced to rare occurrences and no longer necessarily something to fear. ☺

Further Reading

- "CVS II: Parallelizing Software Development," Brian Berliner: <http://docs.freebsd.org/44doc/psd/28.cvs/paper.pdf>
- "Streamed Lines: Branching Patterns for Parallel Software Development," Brad Appleton et al.: <http://www.cmcrossroads.com/bradapp/acme/branching/streamed-lines.html>
- "Writing Version Controllable XML": <http://www.oracle.com/technology/products/jdev/tips/duff/teamxml.html>

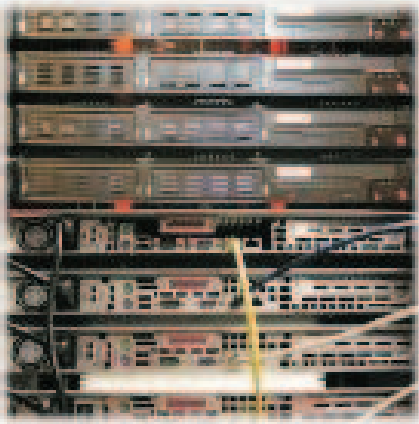
Complex J2EE Hosting made easy.

WebAppCabaretsm

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

\$49
monthly

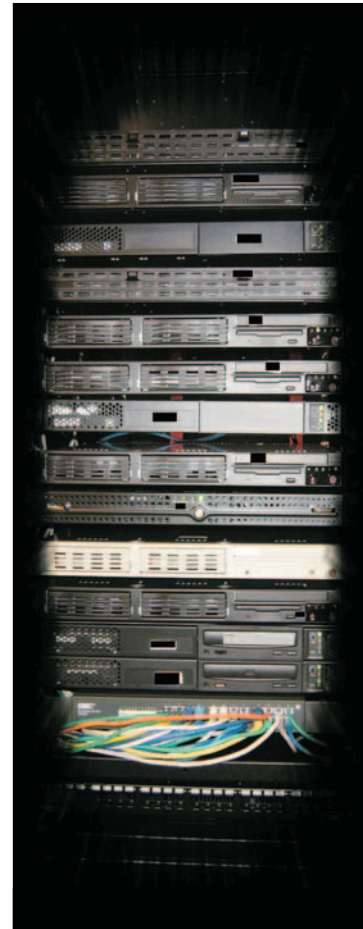
Private JVM
Complete J2EE Hosting solution
for medium size web site.



\$279
monthly

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

J2EE and
Cluster
ready.



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

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

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

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

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

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

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

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

WebAppCabaretsm

Point-and-Deploy J2EE Hosting

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

NGASI
POWERED

Simplify Pattern Matching

by Anant Athale

Use `java.util.regex`

Pattern matching using “regular expressions” can help automate a number of text-processing operations like search and replace, input validation, text conversion, and filters. What otherwise requires significant amounts of code can be done in just a few lines with regular expressions because of the powerful underlying regular expressions processing engine. Some programming languages such as Perl and operating systems utilities such as `grep` have supported regular expressions for a number of years. But before J2SE 1.4, Java (J2SDK) didn't support it and one had to use external packages like Jakarta Regexp, IBM's commercial package (`com.ibm.regex`). Thankfully that changed with the introduction of the `java.util.regex` package. The package provides standard implementations for specifying and handling regular expressions. This article will show you how you can quickly use it to implement regular expressions for pattern-based search features. The article starts out by reviewing some important regular expressions fundamentals and then dives into the details of the package. The embedded examples demonstrate the important constructs through simple use cases.

What's a Regular Expression and Why It's Important

If you've used regular expressions in other languages, the following sections will introduce you to the Java flavor and help uncover some of the new features. If you're not familiar with regular expressions, you'll soon discover how to use them effectively to handle text processing in ways you never thought possible before.

A regular expression is a mechanism to specify a textual pattern and detect the presence of the pattern in a given character sequence. In other words, it's a

pattern language. A regular expressions pattern is typically specified as a combination of two types of characters, *literals* and *meta-characters*. Literals are normal text characters (a, b, c, 1, 2) while meta-characters (ex. *, \$, etc.) convey a special meaning to the regular expression engine discussed in the next few sections. A regular expression engine understands the pattern language. The engine interprets the regular expression, does the pattern match, and processes the results. The language and the engine together make regular expressions a powerful tool that simplifies pattern matching. A given implementation like `java.util.regex` and `JRegex` provides additional query and utility functions (`replace`, `split`, etc.) that are useful in modifying the target text. For details about other Java implementations and implementations available in other languages, please consult the references section.

Meta-Characters

Meta-characters provide advanced expressive power to regular expressions. I will discuss a frequently used meta-character subset that Java supports. For a complete list, please consult the Sun's API documentation (class `java.util.regex.Pattern`). A number of examples that use these meta-characters immediately follow this discussion.

Anchors

An anchor matches a pre-defined position in the target text. Anchors are similar to reference points and are used to determine the relative positions of other elements in the regular expression. They are typically used to match the boundary positions of string, line, word, etc., although they could also match any other position using the special “Lookaround” constructs shown in Listing 1. The

Lookaround constructs match a position based on a given condition. A positive lookahead (`?= Neo`) matches a position that's immediately followed by the text ‘Neo’ whereas a negative lookahead (`?! Neo`) matches the positions that don't have the text ‘Neo’ at the end. Lookbehind constructs (positive `?<=...`, negative `?<!...`) work in the opposite way.

Character Classes, Class Shorthands and Alternation

A character class construct `[...]` is used to specify a list of characters to be included in the regular expression while the construct `[^...]` specifies the character list to be excluded. In the case of `[...]` a match is considered successful if any of the characters specified in the list is found. For example, the regular expression `[cw]` would match the instances of words ‘could’ and ‘would’. The class notation implies a logical OR condition also known as “Alternation” between its elements. Alternation is used to specify conditions `(x|y)` where matching either `x` or `y` is considered a success. Therefore, the earlier regular expression could also be written as `(c|w)ould`.

Special class meta-characters such as `-` can be used to specify a range of values, so class `[a-z]` specifies all letters from a through z. Class *shorthand* is a simplified representation of commonly used classes such as the class digit `(\d)`, word `(\w)`, whitespace, etc. A list of class shorthands available in Java is shown in Listing 1.

Quantifiers

Quantifiers are used to indicate the number of instances of the element (to which they are applied in the regular expression) required for a successful match. Java supports three quantifier types namely greedy, reluctant, and possessive. Greedy quantifiers try to match as much as possible while their reluc-

Anant Athale is a senior software engineer at Motorola Labs. He specializes in enterprise Java technologies and is an active participant in the Java Community Process (JSR 262, 260). He is Sun certified and has a master's degree from Arizona State University.
anant.athale@motorola.com

tant counterparts (with ? at the end) try to match the least required to fulfill a match. What this means is that a greedy quantifier will try to match the entire line whether or not a successful match has occurred. It can turn into real performance overhead when the target text is big. Reluctant (or lazy) quantifiers quit as soon as a successful match occurs without bothering to run through the entire line. Possessive quantifiers (with + appended) are useful in optimizing the match operations since they don't keep the prior match states around. Listing 1 details all three types of quantifiers.

Mode Modifiers

These are special constructs to turn certain powerful regex features 'on' or 'off.' The default mode for these features is 'off' since they involve additional overhead when doing a match. The use of (?i), for example, in a regular expression turns on the case insensitive match mode. Java also supports specifying the mode modifiers at compile time using the static final variables in the class *java.util.regex.Pattern*. The Pattern class is discussed below in the *java.util.regex* section.

Example 1: Input Validation

Let's now review an example that uses the meta-characters discussed so far to address the password validation needs at Zion. The security standards set at Zion Corporation require that passwords contain only alphanumeric characters, with at least one digit and ranging between six and 32 characters long.

Listings 2 and 3 show two possible solutions to the same problem. The first approach (Listing 2) uses the built-in regular expression support inside the *java.lang.String.matches()* method. The second approach (Listing 3) uses the classes provided by the *java.util.regex* package. The underlying mechanics are the same in either case and are discussed next. I'll leave the API specifics to the next section.

Let's see how the solution meets the specified requirements. The regular expression pattern on Line 3 (Listing 2) is same as the *Pattern* pContent (Line 5, Listing 3). The pattern uses a combination of the meta-characters, namely the character class [a-z], class shorthand (\d shorthand for character class [0-9]), and greedy quantifiers (*, +). When

put in a solution context the pattern `"\\b(?:([a-z]*\\d+[a-z]*)\\b"` is successful if between the word boundaries, there are 0 or more letters followed by 1 or more digits followed by 0 or more letters. The mode modifier ?i is used to indicate that the search is case-insensitive. Notice that there are a couple of differences in the regular expressions in the two listings. The obvious one is the use of comments in Listing 3. The other difference is more subtle but important, did you find it? Check out the next section (Capturing, Grouping) to verify the answer.

The pattern on line 4 (Listing 2) addresses the password-length requirement, using the {min,max} quantifier that imposes minimum and maximum limits on the number of successful matches. In this case a match is successful if `"\\b(?:([a-z0-9]){6,32}\\b"` there are between six and 32 instances of alphanumeric characters between the word boundaries. Notice that in Listing 3 the case-insensitive option is specified using the final variables in the class Pattern, which makes the expression more readable. The variables are discussed further in the following sections.



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

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!



Put together, the two regular expressions would accept the passwords “010101” and “m0rpheus” and reject the password “agentsmith.” There are other ways (without regular expressions) one could have achieved the same results but notice how regular expressions make the solution concise and elegant.

More Meta-Characters – Grouping, Capturing

The parentheses `()` are used for two functions, grouping and capturing.

They group the enclosed elements and capture the text matched by the enclosed sub-expression. The backreferences (`\1`, `\2`, etc.) allow the text captured by the group to be used again in the same regular expression. The parentheses are evaluated from left to right and their position (from left) in the regular expression determines the contents of the corresponding backreferences.

The instance of class `'Pattern'` is the compiled representation of the specified regular expression string. The `'Matcher'` object does the match operations on a specified character sequence and provides additional functions to access and use the results from the match. Decoupling the pattern definition from the pattern matcher lets the same pattern be used by multiple matchers. The `'Pattern'` class also provides a static `'matches(String pattern, String text)'` method that can be used in cases where the pattern (and matcher) need not be preserved for reuse later. Notice that in Listing 3 the patterns `'pContent'` and `'pLength'` have been declared outside the method. This lets the pattern instances be reused across multiple invocations of the method and so is more efficient. This option isn't available when using `java.lang.String.matches()` method as shown in Listing 2.

Similarly, there are a number of results query methods. The `group()`, `group(int i)` are normally the ones used most. The `group()` method provides access to the text matched by the previous match application while the `group(int i)` method returns the text captured by the *i*th group (capturing parenthesis). The regular expression `"(\d)([A-C])"` when applied to the string “2140AD,” for example, would make `group(1)` return “0,” `group(2)` return “A,” and `group(0)` return “0A”. Notice that `group(0)` always returns the entire text matched.

Finally, let's discuss some of the text-replacement functions in the `Matcher` class and then look at an example that uses them. The `replaceAll(String newText)` method replaces all instances of the text matched by the regular expression with the new text while `replaceFirst(String newText)` replaces just

“A regular expression is a mechanism to specify a textual pattern and detect the presence of the pattern in a given character sequence”

Java also provides access to the contents of the captured text outside the regular expression through constructs like `'$1'` and `'$2'` where `'$1'` is a handle to the value contained in `'\1'` or the `group(n)` function in the class `java.util.regex.Matcher`.

The other type of parenthesis (`?:`) known as grouping-only parenthesis group but do not capture any matched text. This is a useful construct when backreferences aren't required. It can speed up the match operation by not preserving any data from the match. It also answers the question asked earlier about the difference between the regular expressions in Listings 2 and 3. Notice that Listing 3 uses the non-capturing parenthesis and is therefore more efficient.

Package `java.util.regex`

The package is relatively small consisting of two final classes namely `'java.util.regex.Pattern'` and `'java.util.regex.Matcher'` and an exception class `'java.util.regex.PatternSyntaxException'`. Together these classes form Java's regular expressions framework.

The `compile()` methods in the `'Pattern'` class accept a regular expression string to compile and verify the expression syntax. An unchecked exception `'java.util.regex.PatternSyntaxException'` is thrown if the syntax is invalid. The mode modifiers discussed earlier can be specified as flags at the pattern compile time using the `'Pattern'` class's static final variables (`CASE_INSENSITIVE`, `DOTALL`, etc.). Multiple flags can be specified using the `()` operator as shown in the pattern `pLength` (Line 23, Listing 3).

The `'Matcher'` class allows multiple ways to do a match and query the results. The `matches()` method checks if the input string exactly matches the regular expression and returns a Boolean. The `find()` method checks for an instance of the regular expression in the input string and can be re-invoked to check for multiple instances. These are the two most commonly used search methods although class provides additional methods that are appropriate under special conditions.

the first instance. The advanced replace operations `appendReplacement()` and `appendTail()` together offer fine-grained control over how the replace is done. Their use is shown in the example in Listing 4.

Example 2: Text Conversion

So far we've looked at examples that do validation and determine if the input text meets the requirements. Now let's look at an extension of the previous password example that modifies the target text to make it meet the requirements. Zion's security policy now requires that the passwords be encrypted in some way (fudged) before they're sent across the wire. Let's look at a simple text-conversion utility shown in Listing 4 that uses regular expressions to reverse all the digits in the input text. The idea is not to write a sophisticated encryption algorithm but to demonstrate some of the advanced regular expression features.

The program uses a simple regular expression (Line 5) used to match the instances of the digit class in the password. The digit instances are searched

using the `find()` method, which helps to iterate through the match results (Line 7). The `group(1)` method returns the text captured by the capturing parenthesis(`\d`) in the regular expression, i.e., all instances of the digit class. Each digit is then reversed using the specified array and appended to the new password `StringBuffer`. The `appendReplacement` method takes care of inserting the string found between the matches while `appendTail` takes care of appending the remainder text. The password "010101" would be sent as "989898."

Example 3

A couple of patterns shown in Listing 5 demonstrate the use of regular expressions in matching e-mail and Web addresses. The e-mail pattern (Line1, Listing 5) matches addresses that end with `matrix.com`, `matrix.net`, or `matrix.org`. It also matches the subsequent person name. For example, the pattern matches the word 'Trinity' along with the e-mail address in `tn@matrix.com` (Trinity). The URL pattern matches

the hostname followed by optional path names as in <http://www.zion.com/antimatrix.html>. You'll note the extra 'Pattern.MULTILINE' option argument that indicates that the match should take into account the fact that the URL may span multiple lines. The default behavior just matches the current line.

Conclusion

Regular expressions are handy when writing pattern-matching programs like Internet form validations, converting text to HTML or vice versa, parsing documents, help programs, etc. With the introduction of a regular expressions package in Java, it's become more convenient to use them in a wide variety of applications without having to rely on external packages. We saw some of the commonly used regular expression constructs supported by Java and their use in a number of examples. However, the scope of this article limits the depth and number of constructs discussed. For a more detailed discussion please use the references mentioned at the end.

Note that the regular expression constructs in Java may have a slightly different meaning in other languages that support regular expressions (like Perl, .NET, etc.) and therefore regular expressions may not be entirely portable across languages.

Listing1 shows a list (subset) of regular expression meta-characters that Java supports. Listing 2 shows the Password validation program that uses `java.lang.String`. Listing3 shows the Password validation program that uses `java.util.regex`. Listing 4 shows a text conversion utility using `java.util.regex`.

References

- *JavaDoc J2SE 1.4.2 API*, Sun Microsystems, Inc., (<http://java.sun.com/j2se/1.4.2/docs/api/index.html>)
- *Mastering Regular Expressions*, 2nd Edition, Jeffery.E.F. Friedl, (<http://www.oreilly.com/catalog/regex/>)
- *Java Performance Tuning*, 2nd Edition, Jack Shirazi, (<http://www.oreilly.com/catalog/javapt2/index.html>)



The ODTUG Conference Is Jazzed!

Business Intelligence • Data Warehousing • Methodology • Development DBA
Professional Development • Application Development (J2EE, JDeveloper, PL/SQL, and HTML DB)

For Sponsorship Opportunities
Call 910.452.7444
www.odtug.com

Save \$100
Register by May 30

ODTUG NOW  2005
June 18-22
Sheraton New Orleans Hotel
NEW ORLEANS, LOUISIANA

Reserve your hotel room online at www.odtug.com/2005_conference_location.htm or call 888-627-7033 or 504-525-2500 by May 30 and reference ODTUG for the special rate of \$159/night. A limited number of rooms are available at the government rate on a first-come basis. Artwork by Ivey Hayes, North Carolina Artist www.mccluregallery.net

Listing 1

Anchors:
 \b, \B word boundary
 ^, \A Start of string/line
 \$, \z, \Z End of string/line
 (?=..) Positive lookahead
 (?!...) Negative lookahead
 (?<=...) Positive lookbehind
 (?<!...) Negative lookbehind

Character Classes:
 [...] include a list of characters
 [...] exclude a list of characters

Class Shorthands:
 \ddigit, Shorthand for [0-9]
 \Dnon-digit, Shorthand for [^0-9]
 \s whitespace character, Shorthand for [\n\t\f\r\x0B]
 \Snon-whitespace, Shorthand for [^\s]
 \w word character, Shorthand for [a-zA-Z0-9_]
 \Wnot-word character, Shorthand for [^a-zA-Z0-9_] or [^\w]

Quantifiers:
Greedy:
 ?0 or 1 instances
 *0 or more instances
 +1 or more instances
 {n} exactly n instances
 {n,} n or more instances
 {n,m} between n and m instances
Reluctant: ??, *?, +?, {n}?, {n,}?, {n,m}?
Possessive: ?+, *+, ++, {n}+, {n,}+, {n,m}+

Conditionals:
 | indicates alternation

Mode modifiers:
 iCase insensitive mode (Pattern.CASE_INSENSITIVE)
 mMulti-line mode (Pattern.MULTILINE)
 sDot match all mode (Pattern.DOTALL)
 xFree spacing and comment mode (Pattern.COMMENTS)

Grouping, Capturing
 (? : ...) Grouping only parenthesis
 (...) \1 \2 \3 ... Capturing parenthesis

Listing 2

```
1 public boolean validate(String password)
2 {
3     return password.matches("[a-zA-Z]*\d+[a-zA-Z]*") &&
4         password.matches("[a-zA-Z0-9]{6,32}");
5 }
```

Listing 3

```
1 import java.util.regex.Matcher;
2 import java.util.regex.Pattern;
3 ...
4 ...
5 Pattern pContent = Pattern.compile("\\b # Indicate the
   pattern boundary \n"
6 + "(?: # Grouping only parenthesis for clarity and
   speed \n"
7 + "(?i # Change the match mode to case insensitive
   \n"
8 + "[a-z] # The character class matches any alphabet
   character \n"
9 + "*" # The * quantifier indicates presence of 0
   or more alphabets \n"
10 + "\\d # Match any digits between [0-9]
   \n"
11 + "+" # Quantifier + indicates presence of 1 or
   more digits \n"
12 + "[a-z]* #Followed by any number of characters
   \n"
13 + ")" #End group
```

```
\n"
14 + "\\b #word boundary
   \n"
15 , Pattern.CASE_INSENSITIVE | Pattern.COMMENTS);
16
17 Pattern pLength = Pattern.compile("\\b # Indicate the
   pattern boundary \n"
18 + "(?: #Grouping only parenthesis for clarity and
   speed\n"
19 + "[a-z0-9] # The character class matches any alphanumeric
   character \n"
20 + ")" #End group \n "
21 + "{6,32} #{min,max} quantifier to indicate the match
   limits\n "
22 + "\\b #word boundary \n"
23 , Pattern.CASE_INSENSITIVE | Pattern.COMMENTS);
24
25 private boolean validates(String password)
26 {
27     return
28     pContent.matcher(password).matches() &&
29     pLength.matcher(password).matches();
30 }
```

Listing 4

```
1 import java.util.regex.Matcher;
2 import java.util.regex.Pattern;
3 ...
4 ...
5 Pattern p = Pattern.compile("(\\d)");
6 Matcher m = p.matcher(str);
7
8 private String replaceDigits(String str)
9 {
10     StringBuffer newPassword = new StringBuffer();
11     int iArray[] = {9,8,7,6,5,4,3,2,1,0};
12     while (m.find())
13     {
14         int i = Integer.parseInt(m.group(1));
15         i = iArray[i];
16         m.appendReplacement(newPassword,String.valueOf(i));
17     }
18     m.appendTail(newPassword);
19     return newPassword.toString();
20 }
```

Listing 5

```
1 Pattern pEmail =
2 Pattern.compile("^
3 # Indicate the begin anchor \n" +
4 "From:" +
5 "(\w[-.\w]*.*@matrix\.(?:com|net|org))
6 #Email address match \n" +
7 "\s* # Followed by optional whitespaces \n" +
8 "\(([^()]*)\)" #
9 Followed by the person name \n"
10 , Pattern.CASE_INSENSITIVE|Pattern.COMMENTS);
11
12 Pattern pURL = Pattern.compile(
13 "\b # Indicate the pattern boundary \n" +
14 "(http://" +
15 "\w+(?:[\\.\w+])*\.(?:com|net|org|edu)
16 # Hostname match \n" +
17 "(?:[-\\w:~!@\\$%&*?+,./'])*? # Followed by an optional
   path \n" +
18 "\b # Pattern boundary \n" +
19 ") ",
20 Pattern.CASE_INSENSITIVE|Pattern.MULTILINE|Pattern.COMMENTS);
```




**Why is it that some Java guys
are more relaxed
than others?**

These days, with everything from customer service to sales and distribution running on Java, keeping your enterprise applications available can be pretty stressful.

Unless of course, you've discovered the power of Wily. No other software offers our level of insight. Which means you'll be able to monitor transactions and collaborate with colleagues in real-time. Even more important, you'll be able to optimize performance across the entire application—end-to-end.

So put Wily to work. And keep performance problems from pushing you over the edge.

Get Wily.™

1 888 GET WILY | wilytech.com

wily
technology

©2004 Wily Technology, Inc. The Wily logo is a trademark of Wily Technology, Inc. Java is a trademark of Sun Microsystems in the U.S. and other countries.

Performance in J2SE 5.0

What's new and what's faster

by Osvaldo Pinali Doederlein

If you're a Java developer like me you ask two questions about every major J2SE release. What's new, and what's faster (or slower). Tiger includes a large number of well-publicized, high-profile features like generics, annotations, or the full new API for concurrent programming.

But the adoption of new features usually takes time, and until your J2EE product is updated to the new JRE; your corporate clients' admins let the new JREs pass; large teams get acquainted with new features, or complex systems are carefully redesigned to take advantage of such features. But even in these situations, it's always possible to move to the new J2SE as an optional deployment platform, just for the "free lunch" benefits of better performance or, as in J2SE 5.0, the JVM management features.

My approach here is to look at J2SE 5.0 as an improved runtime for existing apps. I'm sure that all the hard-working people at Sun and the many Tiger JSRs will be mad if developers don't hurry to all those cool new APIs and language features, but using J2SE 5.0 as "just a better runtime" is a good kick-in-the-door to push Tiger – remarkably inside more conservative shops where the rule of "if it ain't broken, don't fix it" drives advanced developers crazy. My personal answer is simple; if it's slow, then it *is* broken, and if the new JVM is faster, then the old *is* slow and therefore *broken*, and so are *all my apps*! But some facts are needed to substantiate this argument.

While J2SE updates always bring new optimizations and performance enhancements, it's also true that major new versions of any software (remarkably dot-zero ones) are often suspect of being "big bloated new version," and Java's no exception. Before proving that J2SE 5.0 is any faster than 1.4.2, we must first show that it's not slower, obeying the Hippocratic rule "First do no harm." So we'll approach

each aspect of Java performance critically, first checking whether Tiger risks making anything worse, and only then looking at the possible gains.

New Features

New APIs are often bigger and more sophisticated than their predecessors (like Swing in Java 2 compared to the old AWT). This is what makes people cry, "Bloat!" (Until they can't live without the new API.) The good thing is that you only bother about the resource requirements of a new API, or about its performance, relative to previous alternatives. On the other hand, any performance advantages won't be collected before the new API is used. In short, existing apps shouldn't be affected, for better or worse, if deployed unchanged with the new runtime.

J2SE 5.0 adds some big new APIs, but they are useful to a narrow audience.

The new **instrumentation, management, and debugging/profiler APIs** are good for developing or monitoring scenarios. They have a performance impact, but only if active. If you're debugging, profiling, or monitoring a JVM instance via JMX or SNMP, you're obviously going to pay a price, but otherwise you shouldn't notice any difference because these features are available.

The new **concurrency API**, one of my Tiger favorites, can be used directly by advanced application developers, but most programmers writing business apps will only use this API indirectly once their middleware (like J2EE containers) are updated to exploit the new APIs. The good news is that when it happens, existing applications will benefit from more efficient concurrency (better performance and scalability) without any change.

J2SE 5.0 adds many small APIs, like those for formatting/scanning, but these won't have a significant impact in the footprint or other performance

measures of most applications except perhaps in specialized apps that make heavy use of such APIs – e.g., some ETL tool that's rewritten to use `java.util.Scanner` to parse monster-sized text files.

Unlike the AWT Swing transition, there won't be a rush to port application code to a major new API from Tiger because there are no new APIs that are as broadly appealing. All the new Tiger features that should become "pop hits" are language-level features, even though some of them are supported by new improved APIs.

Generics have no performance (dis)advantages. That's part of the erasure deal; `javac` uses the generic types to do static validations, then discards these types, so the bytecodes produced are the same as usual (including typecasts). Classes and methods supporting generic parameters will grow a bit because `javac` will generate extra signature metadata in the .class files. But there's no impact at all on code that only uses generic types, either in the traditional way (e.g., declaring an `ArrayList`) or with the new generic syntax (e.g., declaring an `ArrayList<String>`).

Annotations tell a similar story. New metadata can be stored in the .class files, and be available through reflection. If you consider the amount of metadata that's already hacked into J2EE descriptors, javadoc tags (especially with annotation-like tools such as `Xdoclet`), and the large number of new JSRs currently in development to define annotations for everything (servlets, EJB, Web Services, JDBC, and so on), it feels like a possible source of lots of space overhead. Fortunately, the Java annotation facility lets these overheads be controlled. The meta-annotation "Retention" ranges from `SOURCE` (zero overhead) to `CLASS` (it only makes classfiles bigger) and `RUNTIME` (annotations are available for reflection inspection, which consumes memory).

Osvaldo Pinali Doederlein

has a master's in software engineering and works as a technology architect at Visionnaire S/A developing J2EE technology-based applications.

osvaldo@visionnaire.com.br

Other language enhancements like **enhanced for, static import, autoboxing, and enums**, have no direct performance impact good or bad. These features are mostly “syntax sugar.” Indirectly, however, these features create some opportunities to change your performance. Autoboxing makes it all too easy to work with wrapped primitive values, e.g., to put integers inside collections. New code can use autoboxing carelessly in situations where one could work a bit harder (e.g., using primitive arrays instead of collections). On the other hand, when autoboxing replaces old code that used wrapper classes, the result should be *more* efficient. Autoboxing doesn’t use wrappers’ constructors (e.g., new Integer(99)), it uses new factory methods (Integer.valueOf(99)) that return cached objects for the most frequent values (like ints from -128 to +128). (You can also invoke these new methods manually.)

Updated Features

Updated APIs are a gray area. They may impact your performance even if you don’t use the new or improved features. Even if you don’t use any new methods, footprint and speed can be impacted when the improvements involve architectural changes to the APIs. There are some “suspects” in this category for J2SE 5.0.

The **Unicode** spec was updated in version 4.0, including characters that don’t fit anymore in the 16-bit char type, requiring a pair of values for some extended characters. APIs for Strings, text formatting, and regex had to be updated for full compatibility with this new standard, breaking some assumptions that we thought cast in stone – e.g., String.length() still returns the number of chars in the string, but not necessarily the number of lexicographical units. This is now only assured by the new String.codePointCount() method. If you’re a heavy user of core string classes (including the new StringBuilder), don’t worry; the existing methods don’t do anything to support Unicode 4.0; that’s why there are new methods for this. But if you’re a heavy user of higher-level text processing APIs, like java.util.regex.Matcher, their performance may have changed even if your app doesn’t use the new Unicode characters, because the latter APIs will need to use the newer String methods anyway to be safe just in case the text contains Unicode 4.0 characters.

On the other hand, the string manipulation API is enriched by StringBuilder, a faster replacement of StringBuffer. If you’re doing any heavy text processing with handwritten StringBuffer operations, *rush* to port that code to StringBuilder. The new class is much faster simply because it’s not synchronized (which is virtually never necessary). Even though Java’s synchronization is pretty efficient, its overhead is still big for very simple methods, which is the case in most methods in the string classes. For compiler-generated code (from string concatenation operations) there’s no need for manual change. Just recompile your code (with -source 1.5), javac will use StringBuilder instead of StringBuffer. Many core J2SE APIs (mostly in the lang, util, text, math, security, naming, and swing packages) are already optimized to use StringBuilder, so the new API should benefit even unchanged apps, and probably compensate the added costs of Unicode 4.0 compatibility.

Swing’s new look&feels may have different performance traits than the old ones, but usually changes in LAFs are positive or little relevance performance-wise.

Overall, these are relatively small areas to worry about. Other major J2SE releases used to have more significant changes of existing APIs. Fortunately there isn’t any revolutionary API update in Tiger. Still, since Tiger *does* include some changes in this category, I won’t be surprised if somebody produces a microbenchmark yelling that half-a-dozen API methods became five times slower or faster in 5.0.

Implementation Changes

Sometimes one has to go back to the drawing board and rewrite a piece of software from the scratch. This doesn’t happen when the old implementation was deficient. It happens when the old architecture won’t accept new implementation enhancements. The most “memorable” event of this kind was Java 2 (J2SE 1.2), when the entire VM was dumped in the sea, and replaced with new code. We got brand new garbage collectors, JIT compilers, native interface, and virtually all the lower-level APIs (like Reflection and I/O) had to be reimplemented. The new architecture was much more powerful, flexible, scalable, evolvable – and of course, bigger, so most Java processes would suddenly consume a few additional megabytes to run the same code. Changes to this category will happen less often in more mature software, so J2SE 5.0 has only modest items in this category.

On the bright side, once a good architecture is laid out, many important implementation enhancements are often possible without major rewrites that risk stability and a new infestation of bugs. So this is also the category where we find most of the “free lunch” improvements, unless of course if you count the development effort of the JVM itself.

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





The JVM's **low-level debugging and profiling interfaces**: The introduction of the JVMTI and the deprecation of the old JVMDI and JVMPI interfaces (to be removed in J2SE 6.0). The old interfaces were always considered experimental and this move was expected. But like I said before, this only affects your performance in debugging, profiling, and monitoring tasks.

Java2D wasn't rewritten, but it got important implementation enhancements. The most exciting is a new OpenGL-based rendering engine that improves performance and unloads the CPU in machines with good OpenGL acceleration. This feature is still experimental; it will take some time (probably until J2SE 6.0) for Sun to work around driver compatibility issues and enable this by default. The other major enhancement is more immediately useful. Java2D is now much smarter about allocating images in video memory, and image-heavy applications may notice a much reduced heap usage and faster operations on images, especially when the app wasn't hand-tuned to compensate for the limitations of previous Java2D implementations.

The **CDS (Class Data Sharing)** is the only other major architectural change. It's not a full new JVM architecture, but it's still a significant change in one important aspect of the VM – how it manages the memory used by executable bytecodes. This is also a performance-oriented update, which should minimize loading time and per-process footprint.

In detail, CDS is the JRE's new ability to share memory used to load its core libraries between multiple JVM processes. At install time, the JVM converts the core APIs (from the 36Mb `rt.jar`) into a 12MB `classes.jsa` file under `jre/bin/client`. This `.jsa` file is much faster to load because (a) it's smaller, (b) parts of the classloading work are precomputed, and because (c) memory-mapped I/O is faster than the conventional file I/O used to read the `rt.jar` file. So CDS provides a good loading-time benefit even before accounting for sharing.

Classes containing mutable static variables can't be shared, so only ~50% of those 12MB are really shared. This should still deliver up to 6MB of memory savings per process, when multiple JVM processes run in the same host. There is

no sharing of classes outside `rt.jar`, or of native code produced by the JIT compiler, so this initial implementation of CDS is still limited.

Benchmarking this is difficult. On a Windows platform, CDS makes your Java processes appear as if they're using *more* memory, since tools like the Task Manager make the process accountable for the entire size of the memory-mapped CDS file (12Mb) – even the shared part is apparently added to the 'Working Set' and 'Private Bytes' of all processes.

The best way to benchmark is by starting many processes and tracking the Free Physical Memory counter. I did this with 10 instances of Swing's Notepad demo. JRE 1.4.2_05 consumed 120MB, while JRE 5.0 consumed 92MB; an average savings of 3MB per process, which is huge for this small program (30% of the original 12MB per-process footprint). This tiny app doesn't exercise a lot of core APIs; more sophisticated apps should get closer to 6MB of savings.

GC Ergonomics – if you have a "need for speed" and lots of time in your hands, look no further than <http://blogs.sun.com/roller/resources/watt/jvm-options-list.html> for some comprehensive documentation of HotSpot's switches. There are more tuning options in your JVM than in some operating systems. But unless you work at Sun, you won't feel comfortable with most of these options. Too many are more useful to the people who build and support JVMs than for those using them. When I'm wearing my benchmarking hat, I love the fact that many of these options are available, but when I'm back at the office preparing an app deployment, I want 'java `ClassName`' to just work and deliver top performance – well, 'java `-server -mxXXXm ClassName`' is more realistic and still reasonable.

It's well known today that advanced JIT compilers like HotSpot or IBM JDK use dynamic profiling to learn which parts of your code deserve some special care (expensive optimizations). Why not apply the same solution to other aspects of the runtime like GC? This is the motto behind J2SE 5.0's recent improvements in automatic tuning. The JVM will "watch" an application's pattern of memory allocation and reclamation, and attempt to automatically tune its many internal

parameters – sizes, thresholds, options – to provide the best performance out-of-the-box without requiring extensive testing or manual provisioning of those options.

Of course, the runtime can't guess everything. For example, games demand smoothness (no noticeable GC pauses), while servers usually favor raw throughput or scalability. Some level of manual tuning will always be important. The solution is to provide higher-level options like Tiger's new `-XX:GCTimeRatio` switch, allowing you to specify how much time (at most) should be spent in GC, with a single switch that mere mortals without a PhD in GC can understand.

Another related improvement is "server-class detection." This kind of idea is so simple, obvious, and useful that it's surprising how long it took for somebody to think of it. If neither '-server' nor '-client' are passed explicitly to the JVM, it will activate the HotSpot Server by default if the machine appears to be a "server." Right now the criteria is at least two CPUs, and at least 2GB of physical RAM. This is nice so Java doesn't run at half-power when inexperienced developers or administrators don't care to configure the VM even with the most trivial switches. Notice that this behavior is valid only for (32-bit) Solaris and Linux platforms. The 64-bit versions of Sun's JRE only implement the Server VM. On the Windows platform, the problem is that power users like software developers or gamers often use server-class systems, so it's better not to mess with the HotSpot Client default. Remember this before removing all the '-server' switches from your launching scripts.

Conclusion

There are substantial performance improvements in J2SE 5.0 to justify an upgrade even if you can't embrace the new APIs just yet that will render your code undeployable on older runtimes. Even better, there isn't any major risk of reduced performance due to rewritten or expanded features. Tiger should prove a trustworthy and performance-enhancing upgrade for application deployment, and this may be the best tactic to promote the new platform in more conservative environments. ☼



INNOVATE

with the Power of Java™

June 27–30, 2005

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

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

In-depth EDUCATION

Evolve your skills in hundreds of expert-led technical sessions.

Real-world INNOVATION

Evaluate proven tools and technologies in the JavaOne™ Pavilion.

Global COMMUNITY

Celebrate the tenth anniversary of Java technology.

Visionary INSIGHT

Hear what the future holds from industry leaders.

Experience a week unlike any other

EXPERIENCE THE 10TH ANNUAL JAVAONE™ CONFERENCE.

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

Save \$200!

REGISTER

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

JavaOne™
Sun's 2005 Worldwide Java Developer Conference™

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



Joe Winchester
Desktop Java Editor



Geeks, Germs, and Software

At a recent presentation given by a software engineer from a very large automotive company, I gleaned some remarkable facts: for a particular car model where the basic price goes up as the livery becomes lusher and the initials on the trunk longer, half of the increase in value comes purely from software. I had assumed that the extra greenbacks in the price came from fancier music systems combined with a cool retro steering wheel and dashboard fascia; apparently it just comes from good old lines of code.

The average top-of-the-line luxury car has close to a gigabyte of software running it, and consequently enjoys the same development issues as any other project producing such volumes of code. One of these issues is the communication between different modules, and a car's software architecture built around layers and subsystem separation reminds me very much of good old fashioned message queues and transport buses, while the irony of the terminology is poetically sweet.

There are some teething problems, however. I'd have assumed the story was apocryphal had it not been told by a car engineer: some autos are built with Bluetooth technology to provide wireless comms between the car's various subsystems. A not-too-pleased owner of said set of wheels one day found his pride and joy wouldn't start when he turned the key. After the tow truck was called and the garage's grease monkeys failed to find a leak or physical problem, the final diagnosis was that it had caught a phone virus. It seems technology had followed nature, where just as the avian flu virus can jump between species, so malicious software designed for one device can inadvertently infect another. Adds new meaning to the phrase, "Honey, I crashed the car."

At the same conference, apart from the fantastic technology being demonstrated, I was struck by the fact that directly afterward another conference targeted at an entirely different technol-

ogy industry was being assembled in the same venue. The booths and accessories being put together for this were noticeably shabbier and had the appearance of "hand me downs" from another era when said alternative profession enjoyed more prosperity and fortune. The thought the image instilled in me was that we, as software engineers, are now in our purple patch when it comes to our industry's health and future potential.

In "Guns, Germs and Steel" (www.norton.com/catalog/spring99/guns-germs.htm) Jared Diamond articulates how societies were really created around the ability to make tools, which in turn led to the ability to divide labor, allowing for farmers on the one hand and elders and armies on the other. This division of responsibility led to better organizational units that were able to conquer neighboring tribes more easily, and ultimately became responsible for the modern day colonization and the growth of civilizations.

In the Industrial Revolution in Europe in the 1900s it was machinery that changed the way societies lived and worked together, and the much-celebrated heroes of the age built things that smoked, choked, and clanked their way along railroad tracks, canals, or spanned ravines. Civil engineering is now a well-respected discipline that universities teach. When I went through college computer science degrees were considered an oddity, now com-sci is a mainstream rite of passage to getting a career in software.

To a certain extent today's super heroes of social change come from the software industry. Love him or loathe him, but Bill Gates was recently given a knighthood by the Queen of England for his services to the technology industry. This could be seen as on par with the industrial heroes of yore who were given similar gongs in their lifetime and then had statues and museums created to celebrate their lives and contribution to social change.

The change that software is able to leverage on our daily lives should never be underestimated. The Internet is a phenomenal example of how a relatively simple technology has just enabled a massive peer-to-the-power-of-peer network of information and communication that affects each of us daily.

I was recently planning a bike touring vacation and was reading a guide book on the various routes. This book's list of preparation tasks included an entry: "Check you have enough camera film." Staring at my digital camera that I take for granted with its ability to record short video clips with sound as well as images, I was reminded of the cartoon in which a daughter, while sitting on her father's knee, asks the question, "Daddy – tell me what it was like when you had to go to a photo lab and wait to see the results of your pictures?" Each time I take my eight-year-old son around the science museum I am not only impressed and in reverence of the great inventions of the last century that shaped our present day machines, I find it a little disconcerting to see the last room of display cases filled with what looks a little like the contents of my house not so many years ago – vinyl records, disposable camera flash bulbs, floppy disks that are floppy, digital watches with red LED displays, and so forth.

If half of the advances in engineering a luxury car, once the bastion of the nuts and bolts brigade, is done with for loops and asynchronous message calls, are software developers the pioneers driving forward technology? As occurred in previous ages, do we also drive society itself and how people operate their lives? Above all, however, we should all feel proud and privileged to exist in an era when such change is occurring, and also slightly humbled by our individual ability to contribute to how lives and the future are organized. To paraphrase a much-quoted prophecy: "It is the geeks who shall inherit the earth." ☸

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

joewinchester@sys-con.com

NETWORLDSM + INTEROP[®]

LAS VEGAS • MAY 1-6, 2005

Network Infrastructure and Services
Wireless
Security
Performance
VoIP and Collaboration
Data Management and Compliance

See it **All in One Place**
ALL SYSTEMS

**350+ Top Exhibitors on
the Exhibit Floor with
8 Targeted Technology
Zones and Pavilions**

**100+ Educational Sessions,
Including 6 Comprehensive
Conferences Revolving Around 6 Key
Themes, 3 Special Interest Days
and 36 Tutorials and Workshops**

**6 Visionary Keynotes
by Leading Industry
Executives**



**NetworkWorld
Survivor Las Vegas**



GO

Visionary Keynotes



John Chambers
*President and Chief Executive Officer,
Cisco Systems*



Hossein Eslambolchi
*President—AT&T Global Networking Technology
Services, Chief Technology Officer and Chief
Information Officer, AT&T*



Scott Kriens
*Chairman and Chief Executive Officer,
Juniper Networks*



Sean Maloney
*Executive Vice President
General Manager, Mobility Group,
Intel*



Andy Mattes
*President and Chief Executive Officer,
Siemens Communication Networks*

Your Source for Building a Better IT Infrastructure



Copyright © 2005 MediaLive International, Inc., 795 Folsom Street, 6th Floor, San Francisco, CA 94107. All Rights Reserved. MediaLive International, NetWorld, Interop and associated design marks and logos are trademarks or service marks owned or used under license by MediaLive International, Inc., and may be registered in the United States and other countries. Other names mentioned may be trademarks or service marks of their respective owners.

Register Today at www.interop.com

Use priority code **MLAHNV41 and receive
\$100 off any educational product.**

Delegates Reloaded: Walking the Path

Using the reflection API

by Michael Birken

The function pointer, a powerful concept in the C and C++ programming languages, has no direct equivalent in Java. No syntax exists to pass the address of a method to a JButton, for instance, that links it with pressing the button. Instead, Java promotes the use of anonymous inner classes, like this one:

```
okButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        startProcessing();
    }
});
```

Back in October 1996, in an attempt to eliminate the need for this bloated syntax, Microsoft introduced an object-oriented method pointer into J++ called a "delegate." Sun Microsystems, citing the delegate as language pollution, sued Microsoft a year later for violating its Java license agreement. The lawsuit successfully maintained the purity of Java, but it also encouraged Microsoft to develop a competing Java-like language, C#. Today, the delegate lives on in C# and the other .NET programming languages that comprise Microsoft's Common Language Runtime.

Had the delegate become part of Java, would Swing programming be easier? More than simply hooking a component to an action, the delegate could invoke methods directly or asynchronously on a worker thread. Could that technique have solved many of the Swing threading headaches that we're faced with today? This article explores these possibilities by using Java's reflection API to restore the delegate concept to Java.

Building a Delegate

`java.lang.reflect.Method` provides a description of a method including formal parameter types, return type, access modifier, declared exceptions, and annotations. It also provides an `invoke()` method to perform an indirect

call; however, unlike a function pointer, `Method` is not bound to a particular object instance. It's part of a toolkit that includes `Constructor` and `Field` for inspecting classes. To use `invoke()` you must pass in a reference to an object instance.

Listing 1 demonstrates how to use it to call `System.out.println()` indirectly. Since the static member variable `out` is an instance of `PrintStream`, Line 4 calls `getMethod()` on the `PrintStream` Class passing in a description of the method that we're interested in. Note, many of the reflection API methods were retrofitted with varargs as of Java 5; previously, `getMethod()` accepted the method name followed by the method parameter types as a `Class[]`. Line 8 passes the string to print along with `out` to `invoke()` on the `Method` acquired in line 4.

To build a `Delegate` class, I joined together a `Method` and an object instance. Compare Listing 1 to the following snippet:

```
Delegate delegate = new Delegate(
    System.out, "println");
delegate.invoke("Hello World!");
```

First, notice the lack of exception handling. Internally, `Delegate` transforms checked exceptions into unchecked exceptions for cleaner code. Second, observe that when `Delegate` is constructed, the `println()` method is not fully described; the parameter types are absent. `Delegate.invoke()` completes the method resolution on the first call using `getClass()` on each argument. Subsequent calls reuse a cached `Method`; however, this technique, although convenient, fails when a null parameter is used on the initial call. An alternative constructor lets you specify the argument class types explicitly like this:

```
Delegate delegate = new Delegate(
    System.out, "println", String.class);
delegate.invoke("Hello World!");
```

`Delegate` seeks out and binds to methods of any access level, including private ones. Instead of using `Class.getMethod()`, which obtains only public methods,



Michael Birken is actively involved in the design and research of emerging trading technologies at a Manhattan-based financial software company. He is a Sun Certified Java Programmer and Developer. He has a BS in computer engineering from Columbia University.

o_1@hotmail.com

Delegate calls `Class.getDeclaredMethod()`. Unfortunately, both of those functions exclude inherited methods. To get around that, Delegate iteratively applies `getDeclaredMethod()` to all the classes in the hierarchy with the aid of `Class.getSuperclass()`. After locating the method of interest, `Method.setAccessible(true)` is used to suppress invocation access checking.

If you're surprised that you can call a private method from outside of an object, recognize that access modifiers were designed for code organization, not for security; they enable you to expose usage method while hiding implementation details. Suppressing access checking is highly beneficial for unit testing. Normally to unit test implementation methods, you have to grant them at least default (package) access; however, Delegate provides an alternative that lets you keep those methods private.

Since static methods are bound to classes, not instances, Delegate provides another constructor, which accepts a `Class`. The following example creates a Delegate to the `Math.cbrt()` function, a static method introduced in Java 5 to compute cube roots.

```
Delegate delegate = new Delegate(Math.class, "cbrt");
double x = (Double)delegate.invoke(8.0);
```

`Delegate.invoke()` returns type `Object`; in this case, the result is first cast to `Double` and then auto-unboxed to a `double` primitive.

Bridging Tiers

Delegate's primary purpose is to serve as a bridge between an event-driven front-end and a multithreaded middle tier. In Swing, when an event is triggered, it does not execute right away. Instead, it joins the event queue (`java.awt.EventQueue`) where it waits along with other events to be executed by the Event-Dispatch Thread (EDT). The EDT pulls events off the queue and services them one-by-one.

An event that requires a significant time to complete – either because it's computationally intensive or because it makes blocking calls – holds up the rest of the queue, causing the user interface to seem sluggish. In fact, since repaint requests traverse the queue as well, expensive events typically cause components to appear as lifeless solid-colored rectangles.

If an event can't be processed in a timely fashion, it should forward the request to a worker thread. That's the reason why Delegate provides `invokeAsync()`. `invokeAsync()` is called just like `invoke()`, but it dispatches a `java.util.concurrent.ThreadPoolExecutor` thread to execute the method. Since it's an asynchronous call, it returns immediately without a return value. To see this in use, here's an indirect call to `println()` on a different thread:

```
Delegate delegate = new Delegate(
    System.out, "println");
delegate.invokeAsync("Hello World!");
```

Now, Swing components aren't thread-safe; they were designed to be accessed exclusively by the EDT. To safely call Swing methods from worker threads, Delegate pro-

Constructors	
<code>Delegate(Object object, String methodName)</code>	Instance method Delegate. Resolution completes on initial call.
<code>Delegate(Object object, String methodName, Class... argumentTypes)</code>	Fully specified instance method Delegate.
<code>Delegate(Class _class, String methodName)</code>	Static method Delegate. Resolution completes on initial call.
<code>Delegate(Class _class, String methodName, Class... argumentTypes)</code>	Fully specified static method Delegate.
Methods	
<code>Object invoke(Object... arguments)</code>	Call Delegate method directly on same thread.
<code>void invokeAsync(final Object... arguments)</code>	Executes Delegate method on worker thread.
<code>void invokeUI(final Object... arguments)</code>	Executes Delegate method on EDT.
<code>Object invokeUIAndWait(final Object... arguments)</code>	Executes Delegate method on EDT and waits for the result.

Figure 1 Constructors and methods of Delegate

vides `invokeUI()`. If `invokeUI()` is called by the EDT, the method is invoked directly. Otherwise, a request to execute the method is placed in the event queue. Since the call is potentially asynchronous, `invokeUI()` doesn't return a value. On the other hand, `Delegate.invokeUIAndWait()` enqueues the request, blocks until the EDT fulfills it, and then returns the result. The following example demonstrates how to obtain a `TextField` value safely from any thread:

```
Delegate delegate = new Delegate(
    textField, "getText");
String text = (String)delegate
    .invokeUIAndWait();
```


COMMON CONTROLS
www.common-controls.com


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:

Lists

Trees

Tabfolders

Menu

Forms

BreadCrumbs

Calendar

Colorpicker

www.common-controls.com

A Listener Delegate

To join Swing events with methods, I combined delegates with a dynamic proxy. The dynamic proxy is an often-overlooked concept that's been available since Java 1.3. Before I discuss it, let's quickly review the proxy pattern.

A proxy acts as a middleman, serving as a stand-in for another object. Typically that other object and the proxy share a common interface. In the case of Remote Method Invocation (RMI), for example, you access a remote object living in a JVM on a different machine across the network via a local proxy. Internally, the local proxy forwards all method calls to the remote counterparts. The proxy effectively creates the illusion that the remote class is accessible locally.

For Swing components, I needed a proxy with the ability to stand-in for any event listener. I could have created a class that implements all of the listener interfaces in java.awt.event and forwards the calls to Delegate.invoke() accordingly, but that would have required way too much typing.

Luckily, java.lang.reflect.Proxy provides a static newInstance() method for generating a proxy class on-the-fly that implements a specified set of interfaces – hence the term “dynamic proxy.” newInstance() requires a ClassLoader, a list of interfaces as a Class[], and a reference to an InvocationHandler implementation. InvocationHandler contains a single method with this signature:

```
public Object invoke(Object proxy,
    Method method, Object[] args)
    throws Throwable
```

All calls on the dynamically created proxy funnel down to InvocationHandler.invoke() where the Method parameter contains a description of the invoked proxy method. It's a simple matter to use it along with args to make a Delegate.invoke() call.

To generate dynamic proxies, I created a factory class called UIDelegate. Listing 2 shows one of its static create() methods. The return type, UIDelegateListener, is an interface that extends all of the interfaces in java.awt.event. UIDelegateProxy is an inner class that implements InvocationHandler.

The create() method accepts an arbitrary number of arguments in groups of four, each corresponding to a method of an event listener interface. A group consists of the object with a handler method, the name of the event method, the name of the handler method, and the tech-

UIDelegate create(Object... params)	A group for each listener method: (1) object/class, (2) listener method, (3) handler method and (4) synchronous/asynchronous call flag.
UIDelegate create(Object object)	Object containing instance handler methods.
UIDelegate create(Object object, boolean asyncUIHandling)	Object containing instance handler methods and synchronously/asynchronous call flag.
UIDelegate create(Class _class)	Class containing static handler methods.
UIDelegate create(Class _class, boolean asyncUIHandling)	Class containing static handler methods and synchronous/asynchronous call flag.

Figure 2 Static factory methods of UIDelegate

nique to call the handler. The last parameter is a boolean: true indicates that the handler is invoked asynchronously and false causes the EDT to invoke it directly. The following example shows how to associate a handleOK() method with a button:

```
okButton.addActionListener(UIDelegate
    .create(this, "actionPerformed",
        "handleOK", true));
```

Dialog Patterns

Switching from the EDT to a worker thread and back again to complete a business task maintains user interface responsiveness, but it breaks up a conceptually linear flow of execution into scattered, disjoint pieces. Consider a menu item that pops up in an options dialog before executing a task. To avoid the threading issues discussed in body of this article you can link the OK button on the dialog to the business logic with Delegate.invokeAsync(). When the task completes, it callbacks a success or failure method with Delegate.invokeUI(). That technique certainly works and often can't be avoided, but whenever possible, strive for a linear flow of execution.

One way to achieve this is to follow the pattern used by JOptionPane.showInputDialog(). showInputDialog() displays an input dialog and it blocks until the user enters a string. A complicated options dialog, as mentioned in the previous example, would return a bean containing a set of fields. We can link the menu item that manifests the dialog to a controller method using UIDelegate and we can configure it to call the controller with a worker thread. The controller provides the linear flow: it shows the dialog, it blocks for input, it validates the input, and it either displays an invalid fields message or it runs the business logic. The show() method of such a dialog must be thread-safe, unlike showInputDialog() which is designed to be exclusively invoked by the EDT. See the show() method of ProgressDialog for an example of how to do this.

The Mandelbrot Algorithm

The Mandelbrot fractal is the intersection of mathematics, computer science and art. It's amazing that something so visually attractive is generated by such a simple algorithm.

The fractal relies on basic properties of complex numbers. A complex number is a mathematical construct analogous to an object with two fields. It's written as the sum $x + yi$, where x and y are real numbers (double types) and i is the imaginary constant defined by the relation $i^2 = -1$. Each coordinate $[x, y]$ represents a point on the complex plane. The magnitude of a complex number is the distance between that point and the origin, computable using the Pythagorean Theorem.

The image is generated by row scanning a rectangular region of the complex plane and assigning each point $C = [x, y]$ a color by repeating the rule $Z_{N+1} = Z_N^2 + C$, where $Z_0 = C$ until the magnitude of Z_{N+1} exceeds 2. The value of N when the loop terminates determines the color of point C . If the loop fails to terminate after a larger number of iterations, then C is part of the Mandelbrot set and that point is traditionally assigned the color black. For those whose algebra is rusty, given $C = [x, y]$ and $Z_N = [a, b]$, $Z_{N+1} = Z_N^2 + C = (a + bi)^2 + (x + yi) = a^2 + 2abi + (bi)^2 + x + yi = a^2 + 2abi - b^2 + x + yi = [a^2 - b^2 + x, 2ab + y]$.

For further fractal exploration, see the references.

The referenced `handleOK()` method must accept an `ActionEvent`.

For those listeners with multiple methods, you only need to specify the methods that you're interested in. For example, `MouseListener` provides five methods, but if you only need to respond to enter and exit events on the EDT, you can do it like this (see Figure 2):

```
panel.addMouseListener(new UIDelegate(
    this, "mouseEntered", "handleEnter",
    false, this, "mouseExited",
    "handleExit", false));
```

Monitoring Progress

To demonstrate delegates in action, I put together a simple fractal image explorer. The application consists of an image window and a progress dialog. When you click on a point of the fractal, the progress dialog in Figure 3 appears, and a worker thread starts to compute a zoomed-in region. The Cancel button lets the user terminate the running computation.

I opted not to use `javax.swing.ProgressMonitor` because it's not modal, it's difficult to get it to appear, and its cancel button causes it to vanish immediately. Instead, I created a `ProgressDialog` class to encapsulate a dialog with a `JLabel`, a `JProgressBar`, and a `JButton`.

`ProgressDialog` monitors the progress of an object implementing `IProgress`, the interface in Listing 4. `ProgressDialog` contains a Swing timer. Every 0.25 seconds, it updates the progress bar using `IProgress.getCurrent()`. If the user presses Cancel, it invokes `IProgress.requestCancel()`, sending out a request that may not be satisfied immediately. The dialog remains visible until `IProgress.isDone()` return true, indicating that either the computation completed fully or it was terminated gracefully by the user.

Note: the implementation of `IProgress` must be thread-safe and its methods must return rapidly. For instance, in the fractal explorer, `requestCancel()` sets a volatile cancel-request flag instead of blocking until the worker thread terminates.

A cycle starts when a mouse click launches a worker thread using `UIDelegate`:

```
UIDelegate.create(this, "mouseClicked",
    "zoom", true)
```

Before computing the fractal, the worker thread calls the non-blocking `ProgressDialog.show()` method. `show()` safely manifests the dialog and kicks off the Swing timer by forwarding the call with `Delegate.invokeUI()`. The timer is linked to `checkProgress()`, shown in Listing 3, using:

```
UIDelegate.create(this,
    "actionPerformed", "checkProgress",
    false)
```

When `isDone()` return true, the optional `ProgressDialog` constructor parameter, `completedDelegate`, is called back on the EDT with the final progress value. If the completed progress is 100%, the image is obtained using `IProgress.getResult()`, and pasted to the window.

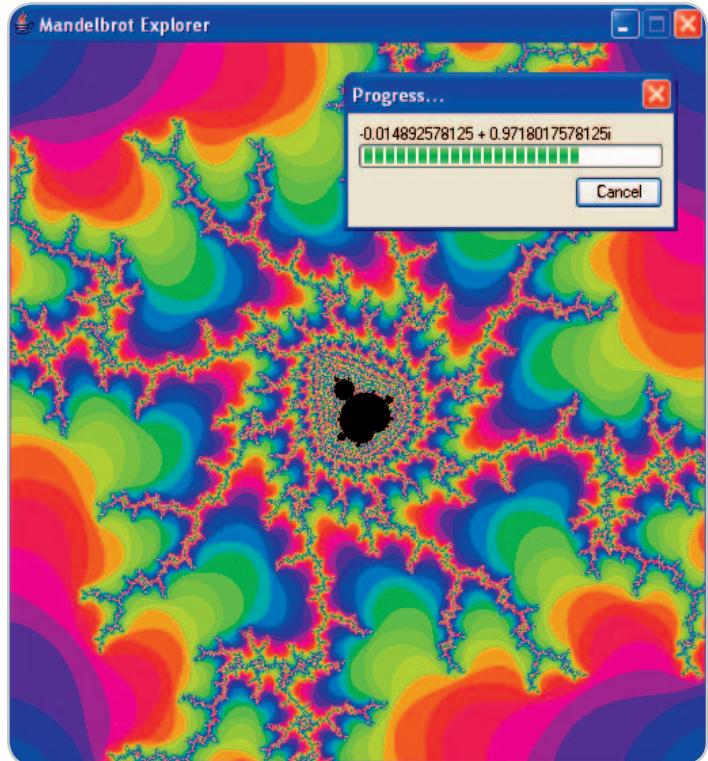
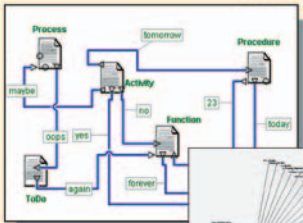
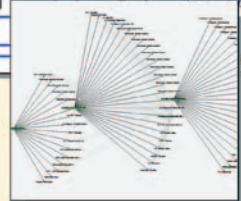


Figure 3 Modal dialog displays image generation progress

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

See the sidebar for details on the image-generation algorithm.

Pros and Cons

Delegates provide cleaner connections between components and handlers both syntactically and thread-wise. The progress monitor example demonstrates how you can focus more on visual design and business logic than on juggling threads. However, Delegate is not type-safe; it's easy to get a runtime exception, for example, after misspelling a function name. Adding a type-safe delegate to a future version of Java might be asking for too much, but what about a method keyword, something like this:

```
Method m = this.actionPerformed(
    ActionEvent.class).method;
```

No need to catch a NoSuchMethodException since the check occurs at compile time.

Also, we're always told to avoid reflection because of its effect on performance. It's conceivable that searching for the Method corresponding to a specified name and a set of parameters takes a performance hit, but once located, what's the penalty of the Method.invoke() call itself? To gain a rough sense the answer, I used System.nanoTime() in a simple test harness like this:

```
long start = System.nanoTime();
for(int i = 0; i < NUM; i++) {
    // ... direct/indirect method call
}
long end = System.nanoTime();
```

```
double average = (end - start)
    / (double)NUM;
```

I ran my tests using Sun's Java 5 JVM with the default settings on a 1.8GHz PC running Windows XP. On my machine, it takes about 3.5ns (3.5×10^{-9} seconds) for a normal method call and about 150ns for a delegated call. Although that's around 43 times slower, it still means that you can make 6.6 million delegated calls a second. Swing applications, at the very least, should be able take full advantage of Delegates without noticeable delays.

You can checkout my performance test harness along with the other code discussed in this article online at www.sys-con.com/java/sourcec.cfm.

References

- *.NET Delegates*: <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/csref/html/vcrefTheDelegateType.asp>
- *Reflection API Tutorial*: <http://java.sun.com/docs/books/tutorial/reflect/>
- *Using a Swing Worker Thread*: <http://java.sun.com/products/jfc/tsc/articles/threads/threads2.html>
- *Explore the Dynamic Proxy API*: <http://java.sun.com/developer/technicalArticles/DataTypes/proxy/>
- *Using Dynamic Proxies to Generate Event Listeners Dynamically*: <http://java.sun.com/products/jfc/tsc/articles/generic-listener2/index.html>
- *How to Use Progress Bars*: <http://java.sun.com/docs/books/tutorial/uiswing/components/progress.html>
- *RMI Tutorial*: <http://java.sun.com/docs/books/tutorial/rmi/>
- *Mandelbrot Set*: <http://mathworld.wolfram.com/MandelbrotSet.html>

Listing 1

```
1 Method m = null;
2 try {
3     Class c = PrintStream.class;
4     m = c.getMethod("println",
5         String.class);
6 } catch(NoSuchMethodException e) {
7 }
8 try {
9     m.invoke(System.out,
10         "Hello World!");
11 } catch(IllegalAccessException e) {
12 } catch(InvocationTargetException e){
13 }
```

Listing 2

```
1 public static UIListener create(
2     Object... params) {
3     return (UIListener)Proxy
4         .newProxyInstance(
5         object.getClass()
6         .getClassLoader(),
7         new Class[] { UIListener.class },
8         new UIDelegateProxy(params));
9 }
```

Listing 3

```
1 private void checkProgress(
2     ActionEvent e) {
3     if (progress.isDone()) {
4         dialog.dispose();
5         timer.stop();
6         if (completedDelegate != null) {
7             completedDelegate.invoke(
8                 progress.getCurrent());
9         }
10    } else {
11        noteLabel.setText(
12            progress.getNote());
13        progressBar.setValue(
14            progress.getCurrent());
15    }
16 }
```

Listing 4

```
1 public interface IProgress {
2     public void reset();
3     public int getCurrent();
4     public boolean isDone();
5     public String getNote();
6     public String getLargestNote();
7     public void requestCancel();
8     public int getMin();
9     public int getMax();
10    public Object getResult();
11 }
```



OSBC

OPEN SOURCE
BUSINESS
CONFERENCE

APRIL 5-6, 2005

THE WESTIN ST. FRANCIS, SAN FRANCISCO
osbc.com | THE BUSINESS OF OPEN SOURCE



osbc.com

THE BUSINESS OF OPEN SOURCE

OSBC is the only conference that focuses on the business of open source strategies and solutions and what they mean to your company. This April, join top level executives and industry luminaries who are paving the way in the Open Source arena.

LEARN to build sustainable, higher-margin Open Source businesses.

HEAR directly from business leaders about how to start and sustain a business on Open Source technology.

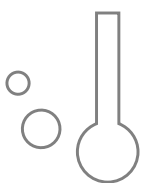
NETWORK with your peers and be the first to meet the hottest up and coming companies in the Open Source industry.

CORNERSTONE SPONSOR



CORPORATE SPONSORS





dotJ JSP Tag Library 2.0.7

Master of tags

Reviewed by
David Castro

The expression “Jack of all trades” ends “and master of none,” but in the case of the dotJ Custom Tag Library produced by dotJ Software, it should end “master of some.” The dotJ tag library is the Swiss Army knife of tag libraries. It provides tags for many different areas of development, from Model-1 form validation to file uploading and text formatting. But it really shows its mastery with its tags for generating sortable grids and calendar widgets.

The dotJ library is a great place to start when planning a new project, especially for those new to JSP. It provides great baseline functionality in handling a lot of different aspects of your site. It has specialized tags for content formatting, e-mail, and forms (including validation), general utility tags for things like code syntax highlighting and file uploading, and presentation tags for making graphically rich grids and calendars.

The library has a free downloadable demo that provides full functionality, but expires at the end of the month. The next demo version is usually out on the first of the month, which means you can continue to demo the library beyond the current month. Installation is almost as easy as any other tag library installation. The only difference is you have to extract a JavaScript file out of the library's JAR and put it in its own directory accessible by the JSP container.

dotJ released the first version of the library several years ago, and some of its functionality has been duplicated in other projects. For example, some of the functionality provided by its content-formatting tags is in the JSP Standard Tag

Library (JSTL). Even in the duplicated areas it's still got a few tricks up its sleeve, such as its handy *titlecase* tag, which will change “jake jones iii, phd” into “Jake Jones III, PhD.” Likewise, its form tags were developed before the wide adoption of the Model-View-Controller (MVC) framework, so its validation functionality isn't necessary on some projects. However, with a bit of work, it's possible to integrate the other features of the form tags, such as tooltips, into an MVC framework. Some of the tags are pre-filled with data, such as the *states* and *countries* tags, which makes them particularly handy.

Target Audience	JSP Web Developers
Level	Beginner
Pros	<ul style="list-style-type: none"> Tags for many areas of development Grid tags alone are worth the purchase price Calendar widget is both beautiful and functional All formatting handled with CSS files
Cons	<ul style="list-style-type: none"> I would like to see it a little easier to integrate into a MVC infrastructure
Platforms	Any platform with JDK 1.2 or later support, and any JSP container that supports JSP 1.1 or later.
Pricing	Professional Edition, \$99. Enterprise Edition, \$599.

Product Snapshot

Id	Last Name	First Name	State	Country	Gender	Employed
1	Jones	Larry	California	US	Male	Y
2	Smith	Amy	Alabama	US	Male <input type="radio"/> Female <input checked="" type="radio"/>	<input checked="" type="checkbox"/>
3	Marshall	Scott	California	US	Male	Y

Update Cancel Delete

Previous Page 1 of 2 Next

The above grid contains 4 records.

Figure 1 Grid created using dotJ library tags

UPCOMING EVENTS

Search:

Select a category:

Thursday, March 3, 2005

06:30 PM [Show entire month](#)

— End of list —

Today

Figure 2 Calendar widget

David Castro has used JSP for four years. During the day he works as a technical writer, but in his off-time he's the lead Web programmer for Cathedral of Praise.

email@davidcastro.com

The library's e-mail tags make it easy to send e-mail from an application, including attachments. I use it to enable a site to e-mail me or the Web master from selected error pages automatically.

You can use IO tags from the library to show the size of the files on a system, do SQL queries, upload files, or automatically color-code JSP source code. You can extend the *showsource* tag by adding color-coding settings for additional languages to a properties file, and use the *showsourcemark* field to show the code on your site without fear of revealing sensitive information.

It provides some nice-to-have utility tags, such as *imagerotator*, *slideshow*, and *watermark*. They rely on the JavaScript that the tags generate on the client side, but I've never had any cross-browser compatibility issues with the ones I've used. A quick look at its JavaScript shows that it tests not just for Inter-

net Explorer and Mozilla, but Safari, Opera, and several others.

Its grid tags make it easy to provide fat-client appearance and functionality without the fat client. You can add image columns and hyperlink columns. If you connect it directly to a datasource, you can enable in-place editing with text boxes, dropdown lists, radio buttons, and checkboxes. The grid has a built-in feature to export the content of the grid to an Excel spreadsheet file.

I have made extensive use of the library's calendar tags, which were introduced in version 2. You can create both fixed and pop-up calendars. Event dates on the fixed calendar are automatically hyperlinked using the *eventlistprovider* tag.

It's easy to update the appearance of all dotJ output to match the color scheme of your site because all formatting is done with CSS and properties files.

dotJ Software is very responsive to customer feedback, and has incorporated my requests on numerous occasions. It has a Web site with forums where you can post questions, and responds quickly to e-mail requests to its support department.

Summary

The dotJ tag library is a great toolkit that covers a lot of different areas for most Web projects. It is particularly well suited to JSP neophytes, but makes life easier for veteran developers too. ☺

dotJ Software

5 Country Lane
Falmouth, ME 04105

Web: <http://www.dotjonline.com>

Test Platform

2.4GHz Pentium 4, 60GB disk, 512MB of memory,
Windows XP with Service Pack 2.

Looking to Stay Ahead of the i-Technology Curve?

Subscribe to these **FREE** Newsletters >

Get the latest information on the most innovative products, new releases, interviews, industry developments, and i-technology news

Targeted to meet your professional needs, each newsletter is informative, insightful, and to the point.
And best of all – they're FREE!

Your subscription is just a mouse-click away at **www.sys-con.com**

IT solutions JOURNAL

NET JOURNAL

JDA DEVELOPER JOURNAL

WebServices JOURNAL

information STORAGE + SECURITY journal

LinuxWorld JOURNAL

LINUX BUSINESS WEEK

wireless JOURNAL

MX developer's journal

WebSphere JOURNAL

wldj JOURNAL

ColdFusion JOURNAL

SYS-CON MEDIA

The World's Leading i-Technology Publisher

Putting Enerjy Software Tools to the Test

From set top boxes to wireless messaging

Reviewed by Alan Barclay

As an independent software developer/consultant starting my career in the early '90s, I was exposed to a large number of different working environments and challenges. My specialty for a while was HP C++ and Motif (often using Interface Architect) on HP-UX, but over the years I migrated towards Sun's C++ and GCC compilers along with X Designer on Solaris. Regardless of the compiler or platform or challenge, there were a couple of tools I always had to have on hand Purify and Quantify (now available from IBM Rational), because they were absolutely invaluable in helping identify subtle problems in C and C++ code.

However, as the '90s drew to a close, I became an independent Java developer, and things like memory analysis, for example, wasn't necessary. Java's got a garbage collector after all, so we don't have to worry about that sort of thing, do we? Some of you are smirking as you read this, because you know the sarcasm with which I wrote that.

Before I let go of my C++ analogies, there was another tool that I carried around with me in my C/C++ days, but this one was a stack of paper rather than a piece of software. It was a document called "Programming in C++, Rules and Recommendations" by Ellemtel Telecommunication System Laboratories, and was an excellent starting point for getting a new team of disparate software developers coding in a standard fashion. Without any intervention, different developers will write code in different styles, leading to delays and potential comprehension problems. Setting coding standards makes it easier for developers to read and understand each other's code.

I'm sharing my past experiences with C/C++ tools with you as a backdrop to a discovery I made a couple of years ago after I forayed into the world of Java development...a discovery that I'll share with you in this product review.

Early in 2002, I stumbled across the Enerjy Software exhibit at a Sun TechDays event in London and was instantly impressed. This article will tell you about my experience using Enerjy's four Java development tools: Enerjy Code Analyzer, Enerjy Memory Profiler, Enerjy Performance Profiler, and Enerjy Thread Profiler. For any Java developer looking to eliminate coding inconsistencies and establish coding best practices, or wanting to combat memory, performance, and thread deadlock issues with Java, this review is for you.

Before I give you the lowdown on the tools, let me describe the development environment in which I work and the sort of Java applica-

tions I write. I use Sun's JDK 1.4.2 with Eclipse 3.0 to write real-time data presentation applications (live data feeds). These apps often involve either an Oracle or Sybase database and/or a Web site. In all cases, I'm working on data presentation, graphics, Java3D, and the like. When I wrote this, I reviewed Edition 6 of the Enerjy tools.

Now on to my review of the tools...

Enerjy Code Analyzer

Enerjy describes Enerjy Code Analyzer as a "best-practices audit tool," but I prefer to think of it in terms of a tool that pinpoints deviations from a "fully configurable" set of rules and recommendations. It's a feature-rich, powerful, low-cost Java code analyzer that I use to rapidly validate the code contributions from the half-dozen Java developers on my project, some of whom are in a different country, making it hard to look over their shoulder as I would if they sat near me.



Alan Barclay holds an honor's degree in electronic and information engineering, and is a hardware and software engineer at Electric Scribe. He has been writing Java applications since the beginning of 1998, and is a Sun Certified Programmer for the Java 2 1.4 platform.

alan@escribe.co.uk

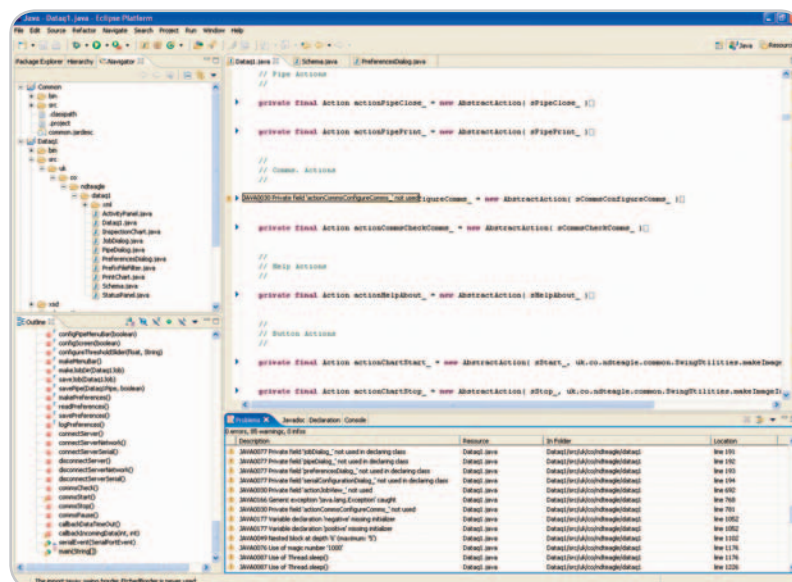


Figure 1 Code analyzer

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

**SYS-CON
MEDIA**
The World's Leading
i-Technology Publisher

Energy Code Analyzer goes beyond pointing out issues with syntax, layout, and style; it's also extremely language-aware. By that, I mean that it identifies omissions and potential errors such as "Switch statement missing default," "Inner class does not use outer class," and "Field hides a superclass field"; mistakes that even the most experienced developer can make. In all, it identifies violations of over 200 Java and J2EE rules and lets you write your own rules for Java code analysis and reporting if there's something unusual that you want it to watch out for.

Energy Code Analyzer integrates tightly with most common IDEs: Eclipse, WebSphere Studio Application Developer, JBuilder, JDeveloper, NetBeans and Sun ONE on Windows, Solaris, and Linux (x86). Alternately, it can be run as an Ant process – as part of a nightly build, for example – producing a text, HTML, or XML report by project or by each developer. Whether looking at the output in report form or highlighting it directly in the IDE (as shown by the tooltip and problem pane in Figure 1), the problems are well described and located for easy identification and resolution.

Code Analyzer's other features include rule prioritization and file filtering.

Rule prioritization lets you prioritize the existing rules as well as the ones you custom-build. And because

of its tight IDE integration, the assigned priority looks just like your IDE's prioritization convention. If necessary, you can choose to fail a build if certain rules have been violated.

File filtering is a neat feature that lets a developer ignore certain code... like auto-generated code that will be overwritten later on. Code Analyzer lets you remove single files or entire packages from consideration.

Whether used by a team leader to check that all team members are obeying the coding standard or used by oneself as a constant reminder that there can be no slacking in professional software development, Enerjy Code Analyzer is great for focusing on quality. Over the years I've come to believe that if you establish coding standards with every development project, you'll pre-empt bugs, save time, and develop more stable and reliable applications. And it doesn't make a difference if you're working in a large team for a big corporation or on your own, making a small application for a client – the need for quality and consistency is the same.

Energy Profilers

I'll break down the individual profiling tools in a minute, but there are some common features shared by all three of Enerjy's profilers that are worth pointing out upfront.

First of all, just as with Enerjy Code Analyzer, all of the profilers

integrate tightly with the IDE. In the case of the profilers, they work with Eclipse, WebSphere Studio Application Developer, and JBuilder on Windows, Solaris, and Linux (x86). Each tool profiles existing launches with minimal configuration from the IDE directly.

All three profilers feature graphical representations of the data and display real-time status views of heap and thread activity while profiling. Finally, the data from the profilers can be exported in tab-delimited or comma-separated format, letting you view, print, or chart your progress.

Enerjy Memory Profiler

Energy Memory Profiler exists to help you understand what's really going on behind the scenes of your memory allocation calls. Integrated with the IDEs outlined above, it can display a real-time status view of heap and thread activity, and it associates heap and object graph artifacts directly back to the relevant methods. Figure 2 shows an example of Memory Profiler with the change in memory usage of a particular application over time.

Do you really know how much memory your application or even a particular method is allocating? Memory Profiler can tell you. By monitoring individual methods, it can tell you how often they're called and the number and size of the objects actually being allocated.

Are you certain that you're not leaking any memory every time you use a certain method? Don't forget that the garbage collector can only recycle memory if nothing is holding a reference to it. Memory Profiler identifies Java memory leaks that survive method boundaries, showing where the object was allocated and how it escaped garbage collection.

A cool feature of Memory Profiler – something I haven't seen in other Java profilers – is something called “on-the-fly profiling.” Basically a developer can change configuration settings on-the-fly – while the application is being profiled – without having to restart the application. Any developer who's had to do this knows how frustrating it is and the time it takes away from the curly braces and semicolons.

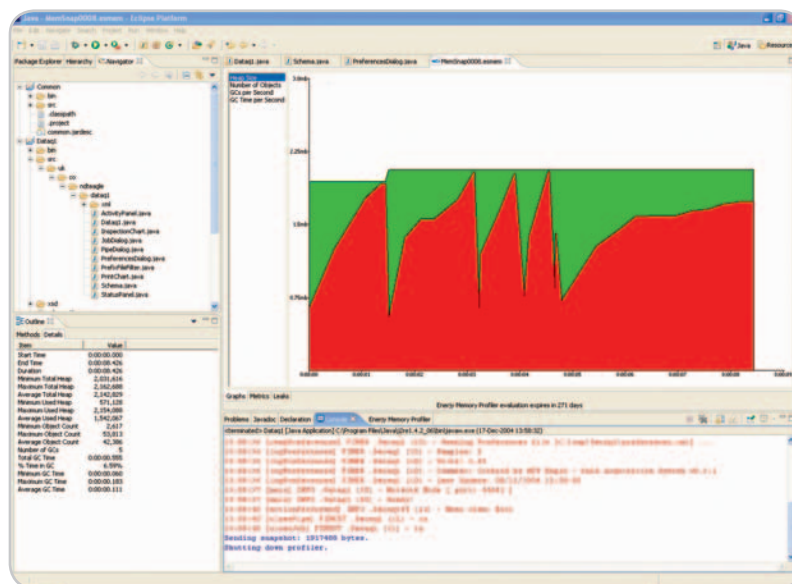


Figure 2 Memory profiler

Advertiser	URL	Phone	Page
Altova	www.altova.com	978-816-1600	4
Business Objects	www.businessobjects.com/dev/p26	888-333-6007	21
ceTe Software	www.dynamicpdf.com	800-631-5006	37
Common Controls	www.common-controls.com	+49 (0) 6151/13 6 31-0	49
DataDirect	www.datadirect.com/jdj	800-876-3101	Cover IV
EV1 Servers	www.ev1servers.net	800-504-SURF	23
Google	www.google.com/jdj	650-623-4000	43
Information Storage & Security Journal	www.issjournal.com	888-303-5282	59
InterSystems	www.intersystems.com/match1	617-621-0600	15
IT Solutions Guide	www.sys-con.com/it	888-303-5282	61
Java Developer's Journal	www.sys-con.com/jdj	888-303-5282	57
JavaOne Conference	www.java.sun.com/javaone/sf	866-382-7151	45
Jinfony Software	www.jinfony.com/jp4	301-838-5560	31
M7	www.m7.com/d7.do	866-770-9770	27
Microsoft	www.msdn.microsoft.com/visual		11
Networld + Interop	www.interop.com	415-905-2300	47
Northwoods Software Corp.	www.nwoods.com/go	800-434-9820	51
ODTUG Conference	www.odtug.com/2005_conference_location.htm	910-452-7444	39
OSBC	www.osbc.com	888-398-1369	5
Parasoft Corporation	www.parasoft.com/products	888-305-0041	7
ReportingEngines	www.reportingengines.com/download/fiere/jsp	888-884-8665	17
Software FX	www.softwarefx.com	800-392-4278	Cover III
WebAppCabaret	www.webappcabaret.com/jdj.jsp	+61 3 6226 6274	35
Wily Technology	www.wilytech.com	888-GET-WILY	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



Another feature that distinguishes Enerjy Memory Profiler from other profilers out there is that it automatically identifies parent leaks, letting you optionally filter out consequential leaks that can be resolved simply by releasing their parents.

The status view function can be activated when a profiling session begins and lets the developer take snapshots, clear collected data, and pause/terminate/resume the application.

Finally, Memory Profiler can profile remote applications including J2EE application servers via a single JVM argument.

In short, Enerjy Memory Profiler is invaluable when it comes to pinpointing Java memory leaks and understanding system memory considerations. The results are often surprising.

Enerjy Performance Profiler

Enerjy Performance Profiler shows you where a particular method in your code is spending its time, both in terms of the actual time spent in sub-methods and the number of calls made to those methods. This information is shown in a number of tables so you can drill down and pinpoint the methods on the critical path. Doing this identifies which methods should be optimized for the best overall improvement in your application's performance.

It shares all of the common traits I outlined above and like Memory Profiler, Performance Profiler integrates seamlessly with the IDE. In fact, if you didn't know it was a separate tool, you'd never know that you were working with something other than your IDE.

Performance Profiler displays real-time status views of heap and thread activity while profiling, and collects both absolute clock time and relative application time for each method. And the good news is that the overhead is minimal.

You can get a quick, at-a-glance view of the worst performance offenders at each method call.

Performance Profiler also shares the same cool feature that Memory Profiler has – on-the-fly profiling. Again, the developer can change configuration settings on-the-fly – while the application is being profiled – without having to restart the application. A real time saver and a revolutionary approach to Java profiling.

The status view function can be activated when a profiling session begins, displaying current (real-time) total/used heap sizes and the running/blocking state of each thread while the application is running. Performance Profiler lets the developer take snapshots, clear collected data, and pause/terminate/resume the application.

Finally, Performance Profiler can also profile remote applications, including J2EE application servers via a single JVM argument.

This tool is a real boon when it comes to focusing your performance optimization efforts.

Enerjy Thread Profiler

Once again, Enerjy Thread Profiler is no different to Enerjy's other profilers in terms of low runtime overhead, reporting capabilities and tight IDE integration. In fact, when it comes to IDE integration, they've done a fine job of fooling you into forgetting that you even have the tools running.

Enerjy Thread Profiler pinpoints each Java thread deadlock when it occurs, identifying all participants and the code that caused the deadlock. It displays a real-time status view of heap and thread activity while profiling and lets the developer take snapshots and pause/terminate/resume the application.

Thread Profiler also gives you a quick, at-a-glance view of monitors by thread or threads by monitor, and it can

profile remote applications, including J2EE application servers via a single JVM argument.

With comprehensive graphs outlining all Java thread processing and monitor activity, Thread Profiler gives you a better understanding of your application to find and fix Java thread problems or fine-tune monitor usage.

Drawbacks

I would be doing you no service if I didn't call out some of what I see as the tools' drawbacks.

First, I'm not a real fan of Enerjy's documentation. It's very pretty (above average, in fact), but I expect that from a quality software company these days. Where it falls short is on the content. For example, I found that I could read the 60-or-so pages on Memory Profiler and feel that I had learned virtually nothing. Lots of words and tables and pictures, but little information. I want more background content and in-depth product information.

The other major gripe I have with the tools is that they seem short on options. Don't misunderstand me: they're not short on functionality, but I sometimes felt as though I might be missing something...as though the GUI hadn't communicated to me clearly.

Java Tools You'll Use

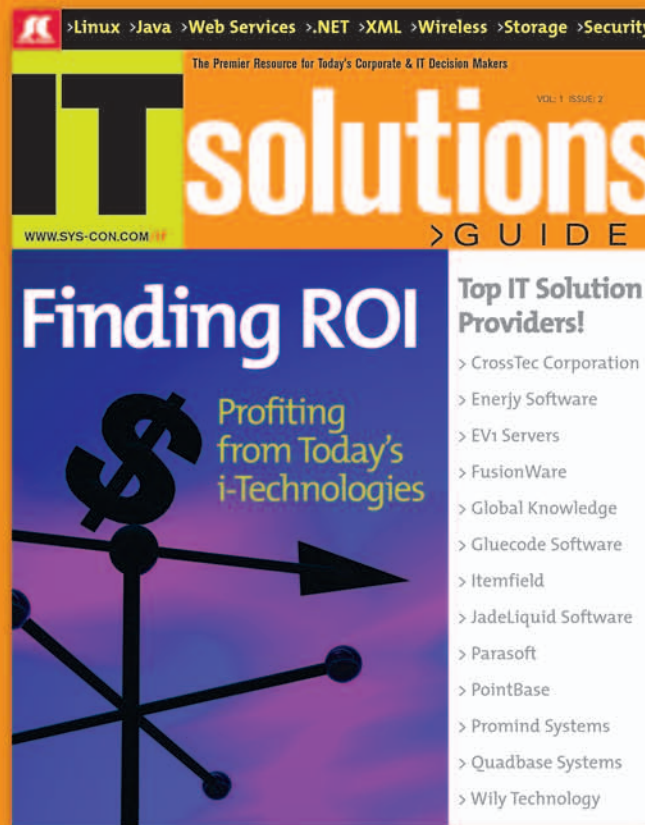
The company likes to say that they create tools you'll actually use (as opposed to "shelfware"). I tend to agree with them. You should have no doubt that these are serious debugging tools for software developers at the highest level, and are not for simpletons who don't know how to program. No matter how experienced a software developer you may be, I challenge you to find standards-, memory-, performance-, or thread-related problems in Java code in a faster or more cost-effective way than using these tools. ☛



If you establish coding standards with every development project, you'll pre-empt bugs, save time, and develop more stable and reliable applications”

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

Evolving the JCP Program Within the Java Ecosystem



Aaron Williams

Keeping the community operating effectively

In my mind, an ecosystem conjures up a green, lush rain forest. The Java ecosystem, like a rain forest, is excitingly complex and able to sustain a diversity life and growth. At the JCP we have successfully balanced a variety of participants, who both compete and cooperate for success within our ecosystem.

The Java Community Process (JCP) program is a vital piece of the vibrant Java ecosystem, where the participants come together to standardize the platforms and APIs. Far from being the whole ecosystem, we're effectively doing our part to create broadly adopted standards with built-in compatibility, and we're working to stay in synch with the rest of the ecosystem. The JCP has been around for six years, and our success over that time has paralleled the success of the rest of the ecosystem—we're growing, getting smarter, and working together.

At the core of the JCP program's success is our ability to evolve with the ecosystem. Who can remember a time before we guaranteed the ability to create independent implementations, or before there were Executive Committees (EC) making the key decisions for the JCP program? Only the gray-hairs remember the days when Sun was the only company allowed to lead Java Specification Requests (JSRs) or when individuals weren't allowed to join in equal standing with the rest of the community members. We've come a long way and we're meeting the needs of the community better than ever.

The most recent evolution of the JCP program happened one year ago. In March 2004 we launched JCP version 2.6, which focused on increased transparency, participation, and efficiency (time-to-market) for JSRs. One year in the life of an ecosystem such as a rain forest is no time at all, but in the Java ecosystem, one year is a long time, and it allows us to evaluate the changes we have made and gauge their effectiveness.

The first area of focus for JCP 2.6 was the transparency of the JSRs. We heard from the community that a couple of changes would be beneficial in this area: giving more people access to early drafts of the JSR specifications, and ensuring all JSRs were optimally transparent for people not participating in the day-to-day activities of the JSR. These JCP 2.6 changes have had a positive impact: Spec Leads are getting significantly more feedback and more useful comments at the first review period, community members are getting access to more open issues and background on decisions made by JSRs, and the public has doubled their opportunities to review JSRs. JSRs under JCP 2.6 are more transparent, and that has proven to be a benefit to the community.

The second area of focus for JCP 2.6 was increased participation. The community wanted more tools to operate JSRs and more opportunities for the public and community to review drafts of the specifications. Since we implemented version 2.6 one year ago, we have seen the number of JSRs led by community members other than Sun continue to rise. By the end of 2004, the rest of the community was leading more than twice the number of active JSRs than Sun. This demonstrates a healthy increase in the investment of the community and the standards piece of the Java ecosystem.

The third area of focus for version 2.6 was increased efficiency, enabling JSRs to get to market faster, encouraging broader adoption with increased compatibility. JCP 2.6 changed the timing of the EC

ballots that each JSR must pass. Under previous versions, all JSRs had to pass their second EC ballot relatively early in the life cycle of the JSR. This caused Spec Leads to get anxious about going to the first review with any open issues, and to spend roughly 200 days getting the JSR ready for that first review and ballot. Under version 2.6, that EC ballot is later in the process, resulting in an average of 125 days for JSRs to get to the first review period. This shows an encouraging trend of JSRs getting more efficient, enabling them to get to market faster and secure broader adoption within the ecosystem.

JCP 2.6 has certainly been more successful than we envisioned when we rolled it out a year ago. This evolution strengthened and grew the Java ecosystem. The participants continue to get more diverse, and there is a need to ensure that the standards for Java technology remain strongly committed to compatibility. The JCP program continues to look for ways to stay ahead of the evolutionary curve.

The Program Office and members of the EC are looking at ways to keep the community operating effectively and efficiently within the Java ecosystem. We are actively looking for input. If you have ideas for solutions to problems that you feel are preventing the JCP program from being as successful as it could be, or if there are things that are discouraging your participation, contact me directly at aaron@jcp.org. Our part of the Java rain forest, I mean ecosystem, is yours to help us evolve. ☺

New Format

Starting with this issue I'm introducing a new format for the monthly **JSR Watch** column, inviting folks involved with the community to share their thoughts and opinions. It's timely that my first guest is Aaron Williams, since this spring the JCP program is celebrating the one-year anniversary of JCP 2.6. Aaron led the JSR that defined JCP 2.6 and worked closely with the community to develop and implement this process specification.

—Onno Kluyt, Chair, Java Community Process (JCP)

Aaron Williams is the manager of the Java Community Process (JCP) Program Office, Sun Microsystems.

aaron@jcp.org

SoftwareFX

Zero To Chart

In Less Than An Hour!



9:04 am

To learn more about the Chart FX products visit www.softwarefx.com and decide which one is right for your platform and target. Then download the 30-day trial version or the Chart FX for Java Community Edition which is a FREE, non-expiring, full development version.

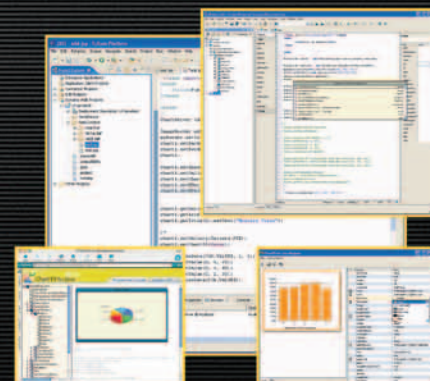
9:07 am

After download, simply install the product appropriate for your needs, platform and target environment.



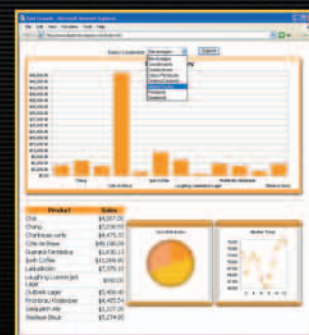
9:13 am

Open the **Chart FX for Java Designer** to get started with the look and feel of your chart. Use your favorite IDE to add functionality and to populate the chart by connecting to your data source. The **Chart FX Resource Center** is also available to help with any questions you may have. Then deploy to your specific application server. ►



9:58 am

Your application is then displayed with an active chart or image that makes any data look great!



Ready... Set... Download!



US: (800) 392-4278 • UK: +44 (0) 8700 272 200 • Check our website for a Reseller near you!

www.softwarefx.com

©2005 Software FX. All rights reserved. Chart FX is a registered trademarks of Software FX, Inc. All other brands are owned by their respective owners.

Remember the good old days?



(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.