SYS-CON

PROTECT YOUR JAVA CODE FROM SECURITY ATTACKS!

See page 44

JDJ

APRIL 2004 VOLUME:9 ISSUE:4

DAVID SKOK
General Partner,
Matrix Partners

SPECIAL: Investing in 'Professional Open Source'

The 'United Nations' of the *i*-Technology World?

EXCLUSIVE INTERVIEW: Dale Fuller of Borland Speaks Out

Plus...

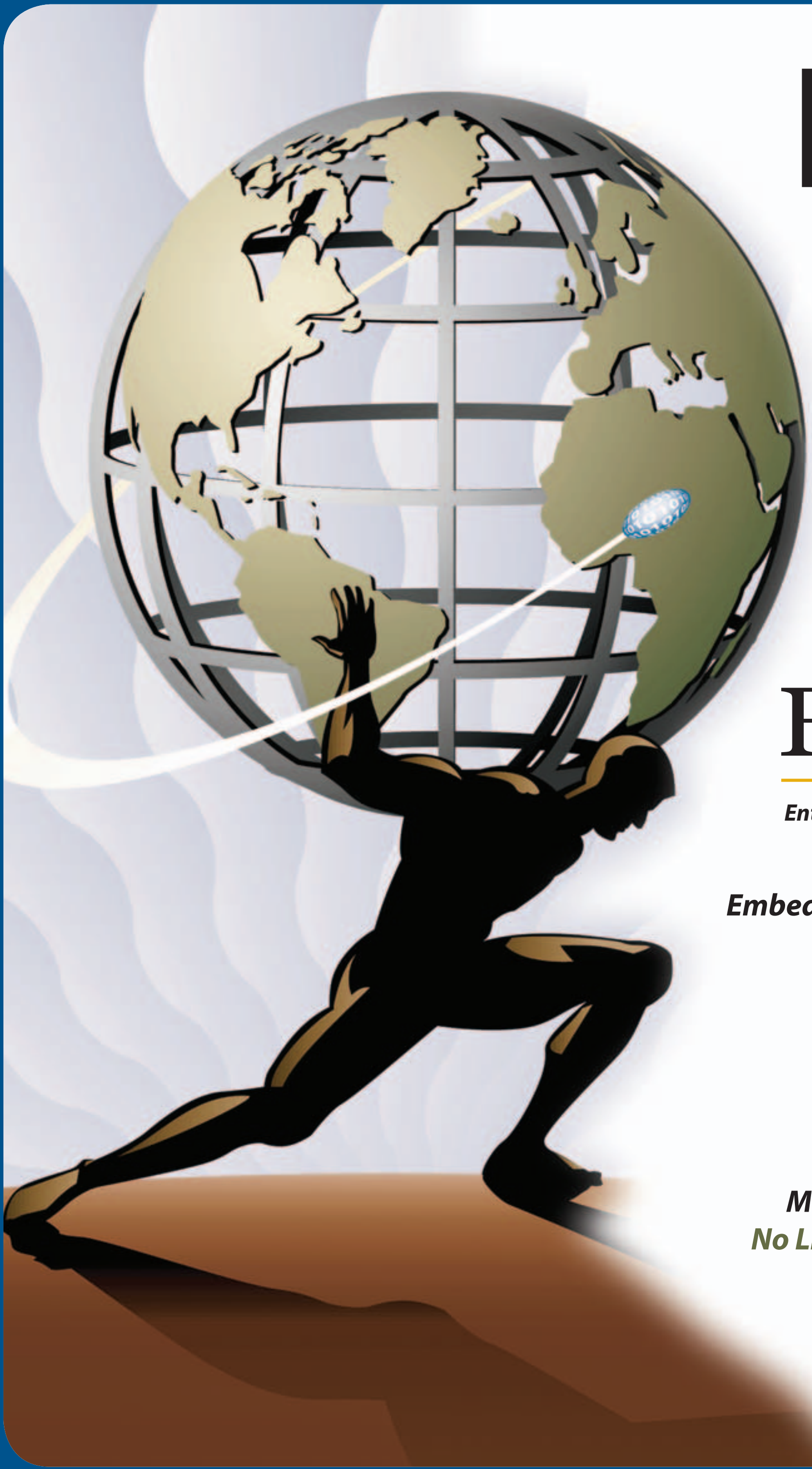
Feature: Escaping Swing's Limitations

Techniques: The Perils of Copy-Paste Coding

JDJ No.1!
HIGHEST DIGITAL CIRCULATION IN THE WORLD!
SEE PAGE 63 FOR DETAILS

THE WORLD'S LEADING *i*-TECHNOLOGY MAGAZINE WWW.SYS-CON.COM/JDJ

WILL IT WORK?
SHOULD YOU EVEN HAVE TO ASK?



BUILD JAVA REPORTS WITH NO LIMITS

FORMULA ONE®

Enterprise Reporting Toolsets for Java Projects & Portals

Embeds Into and Leverages J2EE Environments!
No Limits on IDEs or Application Servers!

UNLIMITED *productivity*
UNLIMITED *data access*
UNLIMITED *report output*

Most Flexible License! Extraordinary Value!
No Limits on Savings for Development Teams!

UNLIMITED *end users*
UNLIMITED *projects*
UNLIMITED *server CPUs*

PRODUCTIVE DEVELOPERS • SUCCESSFUL PROJECTS • EMPOWERED USERS

ReportingEngines™
JAVA REPORTING TOOLS FROM ACTUATE

FREE 30 DAY TRIALS

WEB: http://www.ReportingEngines.com
EMAIL: sales@ReportingEngines.com
TEL: 913-851-2200 • 888-884-8665
FAX: 913-851-1390

FORMULA ONE E.REPORT ENGINE (EMBED PDF, HTML, DHTML, AND XML REPORTS IN JSP AND SERVLETS)
Build reports against Java objects, JDBC, and XML. No report server to set up or maintain.

FORMULA ONE E.REPORT ENGINE FOR WEBLOGIC WORKSHOP (ACCESS LIQUID DATA AS A DATA SOURCE)
All the power of the regular e.Report Engine as a fully integrated BEA Workshop Extension.

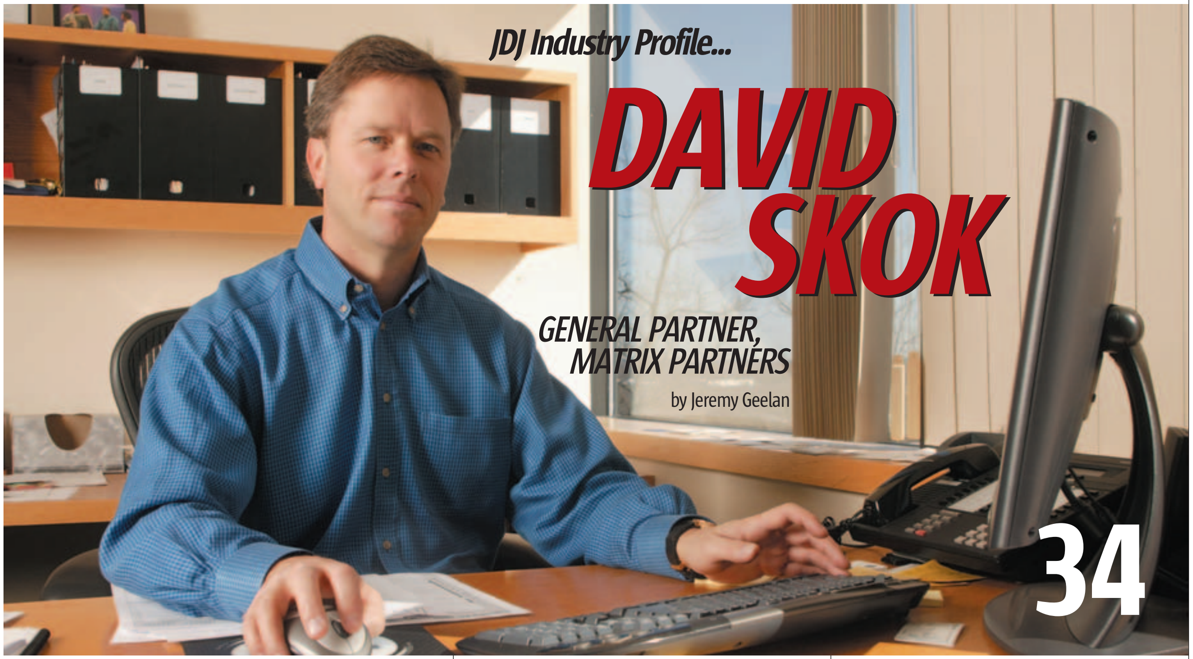
BEA PORTAL REPORTING SOLUTIONS (POWERED BY THE FORMULA ONE E.REPORT ENGINE)
Visually build reports against BEA Portal Event and Behavior Tracking Data.

FORMULA ONE E.SPREADSHEET ENGINE (100% PURE JAVA TOOL FOR BUILDING FINANCIAL APPLICATIONS)
Excel reporting. Excel-like data grids. Server-side calculation and business rules engine.

JDJ contents

APRIL 2004 VOLUME:9 ISSUE:4

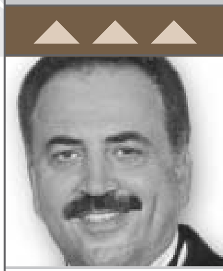
Cover Story: Exclusive



Special Feature

Features

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.



Fuat Kircaali

Where Are They Now?

Summer of ’99

The first time I read Mike Wilson’s book, *The Difference Between God and Larry Ellison: *God Doesn’t Think He’s Larry Ellison*, during the summer of 1999, technology IPOs and dot.coms were at their peak, not to mention Greenspan’s irrational exuberance.

I read through half the book one night, then placed a bookmark at the spot where the founders of Oracle – Larry Ellison, Bob Miner, and Bruce Scott – were moving into their first Oracle office, and Ellison and Scott were dashing a hole in the wall to get the wires through.

We had three editorial meetings scheduled the next day at SYS-CON. Our first meeting was with a newly formed company called PointBase. Our guests arrived on time and we exchanged business cards. When I looked at the names on the business cards, I realized the first one read Bruce Scott, founder and CEO, PointBase. Bruce and I both looked at the book lying on my desk. I said Bruce Scott, as in the cofounder of Oracle? He smiled and said yes. We talked about the broken wall a little bit before we continued with our meeting. :-)

I had the privilege of meeting with and hosting some of the makers and shakers of the software industry that summer. Java was hot and ***JDJ*** was right in the middle of the action. CNBC was “on” 24 hours in our office; more than half of the SYS-CON team was busy day trading. We watched Wall Street legend Harvey Houtkin, the chairman and chief executive officer of a day trading firm, on CNBC for several weeks after nine of his employees were killed in Atlanta.

During the summer of ’99, *JDJ* was in its fourth year of publication. I recall receiving a phone call from the youngest contributing writer of our first issue; he told me that he was taking his Web site public and asked if we wanted to partner with him. As much as I tried, I couldn’t take his call seriously but we saw him on CNBC soon after. His then publicly traded Web site was worth as much as $600 million in the summer of ’99.

Another memorable meeting that summer was with SilverStream Software. Back then we were in a small, not-easy-to-find Irish town – Pearl River, New York. Our meeting took place not even one week after their very successful IPO. We met with the founder of the company, David Skok. My first enthusiastic question to him was “How was the road show, David?” I recall him saying “the road show was very exciting but exhausting at the same time. We did presentations in Zurich, London, New York, and San Francisco on the same day.” (I *think* he said “the same day.”)

I don’t know why but the summer of ’99 reminds me a lot of the *Summer of ’42*, my favorite movie during my teenage years. All I know is that the next summer would be nothing like the summer of ’99 and it hasn’t really gotten much better since then.

On a Friday morning, April 14, 2000, I ran into our next-door neighbor, Jack Martin, at the coffee shop under SYS-CON’s offices. “How is the IPO coming along?” I asked. Jack answered “The party is over. That’s it. Done. Ce fini.” I didn’t realize that Wall Street would experience its biggest one-day fall in history, ending a week in which U.S. markets lost $2 trillion in value — the equivalent to Germany’s entire economy. Worst hit was NASDAQ, the stock exchange favored by high-tech companies such as Microsoft. Bill Gates saw his personal fortune drop $30 billion in a few hours…and Amazon.com and other famous e-commerce companies started laying off staff.

So“where are they now?” Well, I stayed in touch with Bruce Scott, and consider him a friend. I don’t know what happened to our youngest contributing writer; I hope he kept all or at least some of his money. I became good friends with my neighbor Jack Martin and a good neighbor of Wall Street legend Harvey Houtkin. I was wondering where David Skok was until we read about his $10 million funding of JBoss last month and found him. :-) I’m sure you’ll enjoy Jeremy Geelan’s exclusive interview with David in the pages of this issue.

JAVA DEVELOPER’S JOURNAL

Editorial

Editor-in-Chief:	Joseph Ottinger
Editor-at-Large:	Alan Williamson
Executive Editor:	Nancy Valentine
Java Enterprise Editor:	Kirk Pepperdine
Desktop Java Editor:	Joe Winchester
Gaming Editor:	Jason R. Briggs
Contributing Editor:	Ajit Sagar
Contributing Editor:	Glen Cordrey
Contributing Editor:	Jason Bell
Founding Editor:	Sean Rhody

Production

Production Consultant:	Jim Morgan
Associate Art Director:	Tami Beatty
Associate Editors:	Jamie Matusow Gail Schultz Jean Cassidy Jennifer Van Winckel
Assistant Editor:	Kelly Flynn
Online Editor:	Lin Goetz
Research Editor:	Bahadir Karuv, PhD

Writers in This Issue

Gina Assaf, Klaus Berg, Ted Farrell, Jeremy Geelan, Bob Griswold, John Hutton, Onno Kluyt, Adam Kolawa, Stuart Lawrence, Peter Leitner, Tilak Mitra, Paul Mukherjee, Joseph Ottinger, Kirk Pepperdine, Paul Perrone, Stefan Piesche, Roberto Scaramuzzi, David Shay, 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 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.

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

WebRenderer™

Standards compliant embeddable web browser component

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

Why WebRenderer?

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

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

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

JadeLiquid™
Software

Joe Ottinger
Editor-in-Chief

Looking for **Instant Solutions?**

There is no magic bullet. Managers and developers alike have a tendency to look for a simple, one-shot solution to address a series of complicated issues, even while we all acknowledge that there is no philosopher's stone. That fails to stop us, though – the search continues for some mythical fountain of ability (located in Florida or India, surely) against all applications of reason and sanity. Unfortunately, there's no replacement for actually rolling up your sleeves as appropriate for your current project.

In the end, every magic solution fails under the stress of reality: no project manages to fit itself into a given solution. It's the solution's responsibility to be flexible enough to compensate for the problem space.

Personal bias is inevitable, unfortunately. It's unrealistic to remove the developer from the solution, and just as unrealistic to pretend that the biases don't affect what the developer creates. Someone who's successfully used Hibernate for persistence tends to always think Hibernate's appropriate until shown otherwise, often through drastic failure. Likewise, someone who's had a failure with EJB tends to feel that EJB is inappropriate, even after being assigned to a project for which EJB is a good match. Few developers are able to correctly anticipate when conditions have changed in such a way as to controvert their own experiences.

Just as solutions aren't able to magically fit themselves into a problem, developers aren't able to do so either. Very rarely is a given hire – whether it's of a single developer or a group of developers – going to correct a project's problems, or yield a solution that involves no commitment. Even more rarely is a purchase going to make that kind of difference.

In my opinion, the best the industry has in terms of great solutions are tools like IDEA, Ant, Optimizeit, and a few others. The hallmark of all of these is that they are not solutions themselves. All are related to the creation of tuned solutions. Even more, none fit into the "popular product" mindset occupied by APIs and products like Hibernate or EJB, AspectJ or XDoclet, Struts or WebWork. All are one layer removed from those products, which work…but fail in the "magic bullet" category.

Solutions tend to be general in nature, such as "manage a bank account" or "process a new policy." These are amorphous concepts, things with potentially very complex use cases (especially for general use, as you have to anticipate how all users will vary processes). As a result, it's horrifically difficult to get all of the nuances right, and I think even casual users can detect the assumptions made by developers and analysts.

The products I like are luckier; they're much simpler. Note that I didn't include products I use all the time in varying capacities: Eclipse isn't there because it's trying to be a platform and not just an IDE, and it shows – jack-of-all-trades, master of none. Same for NetBeans, with more emphasis on the platform side of things. JBuilder is an excellent development environment…but it, too, suffers from Borland's mindset, although parts of the suite – such as Optimizeit – are excellent and highly recommended.

The final result is that no one is going to able to solve everything without putting in time and effort. No product will suddenly solve your problems, no matter what the advertising (or résumé!) says; prior experience simply isn't valid as an indicator for current issues. What you should look for is a person – not a tool – who understands that every solution is unique and is good at creating solutions, as opposed to someone who spends his or her time flaying the same historical data over and over again. There is no magic bullet – but there are people who are good shots.

Joseph Ottinger is a consultant with Fusion Alliance (www.fusionalliance.com) and is a frequent contributor to open source projects in a number of capacities. Joe is also the acting chairman of the *JDJ* Editorial Advisory Board.

josephottinger@sys-con.com

SYS-CON MEDIA

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 *roybn@sys-con.com*
Director, Sales and Marketing:
Megan Ring *megan@sys-con.com*
Advertising Sales Managers:
Alisa Catalano *alisa@sys-con.com*
Carrie Gebert *carrieg@sys-con.com*
Associate Sales Managers:
Kristin Kuhnle *kristin@sys-con.com*
Beth Jones *beth@sys-con.com*

Editorial
Executive Editor:
Nancy Valentine *nancy@sys-con.com*
Associate Editors:
Jamie Matusow *jamie@sys-con.com*
Gail Schultz *gail@sys-con.com*
Jean Cassidy *jean@sys-con.com*
Jennifer Van Winckel *jennifer@sys-con.com*
Online Editor:
Lin Goetz *lin@sys-con.com*

Production
Production Consultant:
Jim Morgan *jim@sys-con.com*
Lead Designer:
Tami Beatty *tami@sys-con.com*
Art Director:
Alex Botero *alex@sys-con.com*
Associate Art Directors:
Louis F. Cuffari *louis@sys-con.com*
Richard Silverberg *richards@sys-con.com*

Web Services
Vice President, Information Systems:
Robert Diamond *robert@sys-con.com*
Web Designers:
Stephen Kilmurray *stephen@sys-con.com*
Christopher Croce *chris@sys-con.com*

Accounting
Financial Analyst:
Joan LaRose *joan@sys-con.com*
Accounts Receivable:
Charlotte Lopez *charlotte@sys-con.com*
Accounts Payable:
Betty White *betty@sys-con.com*

SYS-CON Events
President, SYS-CON Events:
Grisha Davida *grisha@sys-con.com*
Conference Manager:
Lin Goetz *lin@sys-con.com*

Customer Relations
Circulation Service Coordinators:
Shelia Dickerson *shelia@sys-con.com*
Edna Earle Russell *edna@sys-con.com*
Linda Lipton *linda@sys-con.com*
JDJ Store Manager:
Brunilda Staropoli *bruni@sys-con.com*

Leaner, Meaner, Faster Java Development.

Kirk Pepperdine
Java Enterprise Editor

The Commercialization **of Open Source**

We've all heard the news: JBoss has received $10 million in funding and now it's time to sit back and mull it over. Without a doubt this infusion of capital is a signal of confidence for JBoss Group. But is this investment a good thing for open source? Not an unimportant question for those of us who have decided to use open source in our enterprise applications. If organizations are just now deciding to use open source, this announcement could cause them to rethink their decision and weigh the possibility that their choice may not be so open as it has been. We do have a few exemplars that we can draw from to help us understand what could happen. The most obvious are IBM alphaWorks, the Apache Foundation, and the various Linux vendors, some of whom have IPOed.

alphaWorks is home to a number of open source projects. We don't hear about this IBM-funded effort as much as we used to because many efforts have simply taken a back seat to Eclipse in the media. Another reason is that many of the Java projects have been donated to the Apache Jakarta project. Even so, it continues to act as an incubator while being funded by IBM.

The fact that Apache receives a significant portion of its funding and support from Collabnet is neither highly publicized nor used in marketing efforts. Apache's branding has developed organically. Aside from the number of high-quality offerings (donated by IBM and others), much of the respect that Apache enjoys is due to Collabnet. Being backed by Collabnet has given businesses enough trust in the viability of Apache that they are willing to base critical business applications on Apache technology. Because of this, Apache has been a big win for Sun, IBM, O'Reilly, Oracle, Borland, and others.

What of open source projects that don't enjoy this level of support? Is JBoss a viable option for businesses without the backing of a group such as Apache or the JBoss Organization? Could Linux be where it is today without the efforts of companies such as Red Hat? What does all of this say about the future of open source projects? Will the companies that are supporting open source continue to do so once they face real pressure from investors? Will people be willing to donate their time to open source projects knowing that others will be profiting from their efforts?

There is no doubt that the JBoss project enjoys a large grass roots following. The difference between the JBoss Organization and Apache is that the JBoss founders have been much more vocal about their efforts than Apache has been. In addition to the attention that this loud chest thumping has attracted, it has also made some people nervous about the future of the JBoss project. There is no question that JBoss products will survive. This infusion of capital all but ensures a healthy future for them. The question is: What form will that future take if the principal sponsors of the project are now off trying to satisfy those who provided this capital?

It is conceivable that under the limited GPL license, future development on JBoss will occur in a totally commercialized context. Will we see two versions of JBoss: one open source and another for those who are willing to ante up licensing fees? Will documentation and support only be doled out to those who are willing to pay? Not according to Marc Fleury. Fleury has the personality and attitude that was needed to bring JBoss to this level. He seems to know when to thumb is nose at the establishment and when to pull back. As long as Fleury is in control, I expect he will keep his word and JBoss will remain as it is. The question is: How long can Fleury maintain control?

In an article published on CNet (June 2, 1999) just after Red Hat announced that it had filed for an IPO, an analyst raised the question, "If a company such as VA or Red Hat went public and made a lot of money off of Linux, what does that mean for all those people who've done a lot of work and don't necessarily make money out of it? Will they still want to contribute to Linux?" A quick survey of the Fedora (a project to build a complete OS from free software) IRC channels showed that more than 300 people were signed onto #fedora and 120 where signed onto #fedora-devel. More important, there were people engaging in dialogue while trying to advance the development of this open source project. While these numbers don't rival the thousands that participate in projects like Ant, it still appears to be a healthy vibrant community working toward a common goal.

It's unclear how many of the people who have contributed to Red Hat's success have been compensated for their efforts. The JBoss Group has made a concerted effort to compensate people who have contributed to the JBoss project. The recent hiring of Gavin King illuminates the group's efforts to continue to support Professional Open Source, that is open source software developed by professionals earning a living from the product that they are contributing to. Is this any different than what Sun, IBM, and others have been doing with their support of Apache Jakarta?

Only time will tell if the JBoss Group can successfully transform themselves from a group of developers working on an open source project into a viable commercial enterprise. Or if JBoss as an open source product will remain viable and not be consumed by JBoss, the commercial enterprise. If history has anything to say, they should be able to pull it off.

Kirk Pepperdine is the chief technical officer at Java Performance Tuning.com and has been focused on object technologies and performance tuning for the last 15 years. Kirk is a co-author of *Ant Developer's Handbook* (Sams).

kirk@javaperformancetuning.com

The POWER of Interactive Information

Your users demand to see data in more than just an HTML table. Using EspressReport you can give them interactive, actionable information in the format they desire.

Easily create sophisticaed reports with detailed charts, multiple tables, and drillable data. Users can view reports as HTML, DHTML, PDF, Excel and RTF with zero client.

Designed to leverage your application infrastructure EspressReport embedds fully within your J2EE application server and quickly integates with applications, JSPs and portals.

Built-in ad-hoc reporting capabilities allow your users to create queries and reports themselves, freeing development resources.

With EspressReport you can:

- ✓ Build reports using database, XML, text, or Java object/EJB data with extensive parameter support.
- ✓ Plot data in over 30 2D and 3D charts.
- ✓ Fully integrate reports into your applications using the extensive Java APIs.
- ✓ Provide end-to-end user and data security within reports.
- ✓ Schedule reports with archiving and email delivery.

EspressReport®

Quadbase
Quadbase Systems Inc.

www.quadbase.com
Download a free evaluation today!

(408) 982-0835
sales@quadbase.com

Service-Oriented **Architecture**

Beyond Web services

by Ted Farrell

Ted Farrell is architect and director of strategy for Application Development Tools at Oracle Corporation. He is responsible for the technical and strategic direction of Oracle's development tools products, including Oracle JDeveloper 10g, Oracle's Java and XML Integrated Development Environment (IDE) for J2EE applications, and Web services development.

ted.farrell@oracle.com

Chances are you've heard the term service-oriented architecture (SOA). It describes a software architecture in which reusable services are deployed onto application servers and then consumed by clients in different applications or business processes. If you've tried to find information on SOAs, the chances are also good that you found a description that includes Web services, often exclusively. This might have led you to the conclusion that if you aren't using Web services, you have no need for SOAs. This couldn't be further from the truth.

The problem starts with the definition. SOAs are designed to decouple the implementation of a software service from the interface that calls that service. This allows clients of a service to rely on a consistent interface regardless of the implementation technology of the service. Instead of building big, monolithic applications, developers can build more agile services that can be deployed and reused across an organization for different applications and processes. This allows for better reuse of software functionality, as well as for increased flexibility because developers can evolve the implementation of a service without necessarily affecting the clients of that service.

To this end, the main requirement of an SOA is that the interface to the services is decoupled from the implementation. When you hear this description it might remind you of another technology that is discussed a lot: Web services. Web services allows access to a diverse set of functionality through a standard protocol-and-interface definition. Web services and service-oriented sound like the same thing, but they're not.

SOAs and Web Services

Today's examples of SOAs are mainly in the business process area. These applications are focused on building business process flows where several services are used in conjunction to accomplish a larger task. These processes also require data transformation as different types of data are taken from one service and sent to another service in the flow. In this example, Web services are often used as the services in the process. Web services provide a good mechanism to communicate with a diverse set of technologies that would be much harder to integrate without them; however, a problem arises when a developer needs to add a non–Web service service to a business process that also contains Web services. To handle this case, either the client has to code to multiple interfaces, or the developer needs to add an abstraction layer that shelters the client from the differences in the service interfaces and implementations. This abstraction layer is known as a service-oriented interface (SOI). In the case where all of the services in the process are Web services, the Web service interface is the service-oriented interface.

Web services are one example of an SOI. The problem with using Web services exclusively in a service-oriented architecture is that developers have to wrap any functionality they want to expose as a Web service. This makes sense in cases where the service is built using a foreign technology or is supplied by a different vendor, but ideally you would try to avoid the overhead of packing and unpacking data into XML for services running on the same infrastructure. However, because Web services are one of the few standardized examples of an SOI, they are often assumed to be the only choice when building an SOA.

Beyond Web Services

A non–Web service example of an SOI can be found in the Java Specification Request (JSR) 227. JSR 227 is a standard data-binding and data-access facility for J2EE. The goal of JSR 227 is to abstract the implementation of a data source from a client looking to take that data and bind it to a user interface. Whether you're displaying data from a database, Enterprise JavaBean (EJB), legacy system, Web service, or plain old Java object, the interfaces and data are the same.

This is shown in the "Model Layer" in Figure 1. By providing a consistent interface to any business service, the model layer is implementing an SOI. The functionality provided in any SOI really comes down to the required information. For example, the model layer in Figure 1 doesn't necessarily have to provide full access to all features and functionality of the business service that it represents. Instead it just needs to provide enough functionality for clients to successfully bind that data into user interfaces and then manipulate it.

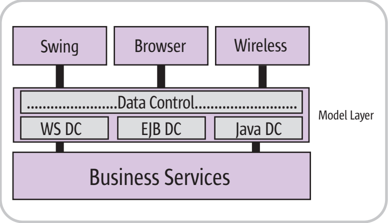


Figure 1 The service-oriented interface

Inside an SOA

Taking a closer look at how this is done leads us to the DataControl. The job of the DataControl in the model layer is to abstract the implementation of the business service from the client, as well as to act as a mediator between the client and that business service. The DataControl API describes certain functionality of the business service that the developer wants to expose. For all of this to work, the DataControl needs to expose this information in an implementation-neutral manner. Regardless of the implementation of the business service, the description is the same.

It's worth highlighting why a DataControl was chosen over a Web service as the interface for the data-binding SOI. Using an SOI that is focused on a specific job dramatically increases its viability, performance, and acceptance. For example, while Web services typically run on the server, the DataControl runs on the client because that's where the data will be displayed and changed. The DataControl can also

represent multiple kinds of services and does not need to package the data as XML in order to send it to the client, as it's already running on the client. Also, while a Web service is designed for calling operations on services, the DataControl can also access attributes, which is a very common way to display data in a user interface. Trying either to change the data-binding architecture or bend Web services in order for them to meet the requirements of the data-binding SOI would have been the wrong decision. Using an SOI that's designed for a specific task allows developers to get the advantages of an SOA without having to sacrifice the design, performance, or ease-of-use to get there.

Let's get back to the DataControl. The DataControl is able to abstract the implementation of the business service by breaking things down into a common form. In this case, that common form is a set of attributes and operations. Any business service has zero or more attributes and operations. Attributes are simple setXX and getXX methods that conform to the standard JavaBean pattern. An operation can be any other method on the business service and can take parameters and return values. The DataControl also describes the data that is returned from an attribute or operation. This description is represented as generic objects with typed attributes. Attributes can be primitives or other objects. This allows for complex data models to be described regardless of where the data comes from.

How It Works

The DataControl processes requests from the client and maps them into the implementation-specific data that it retrieves from the business service. By getting the description of the attributes, operations, and data in a common metadata format, the client no longer cares about the implementation details. The DataControl becomes the contract between the business service and the user interface (client), and the developer has the freedom to change and modify the business service as necessary without having to break the contract.

Figure 2 shows an example of this SOA using an EJB business service. The DataControl sits on the client and communicates with the application server as any EJB client does. When the UI client calls an operation on the DataControl, the DataControl in turn calls the Java Naming and Directory Interface (JNDI) to access the EJB, makes the method call on that EJB, and gets back data transfer objects. The DataControl then passes back a collection of maps describing these objects to the UI client. The UI client can then make get() calls on any of the maps to access the different attributes of the data objects. The get() method of the map delegates to the DataControl, which in turn responds with the actual data from the EJBs. The same is true when the UI client updates the data using the set() method of the map.

Since the DataControl is running on the client, there's no need to repackage the data into XML or JavaBeans. It's all handled in the delegated get() and set() methods of the map right on the client.

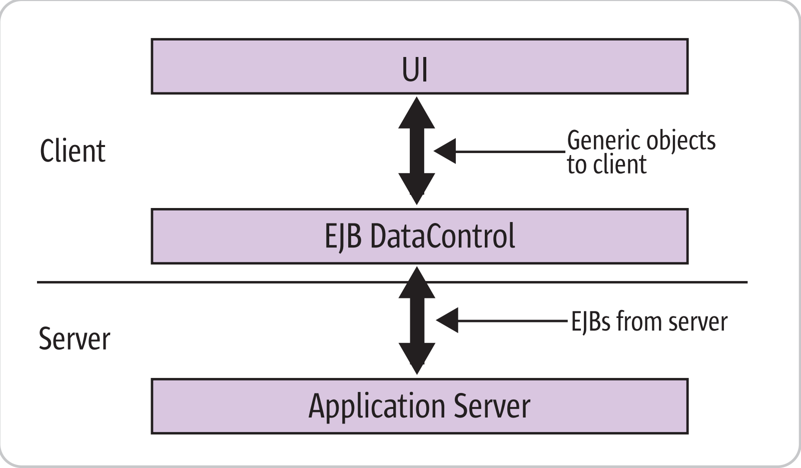


Figure 2 SOA using EJB business service

What's in It for Me?

The DataControl is completely in charge of how the data is cached, loaded, and managed. That functionality is sheltered from the client because of the SOI. This allows developers to optimize and tweak the implementation of their business service or extend the functionality of the DataControl without affecting the client. For example, we could modify the application described by Figure 2 and add client-side caching. In this case, we would extend the DataControl to cache the data on the client, saving trips to the server for each call. The server would then notify the DataControl when the data is invalidated, causing it to make subsequent calls to the server to refresh its cache. All of this can be done without any affect on, or knowledge of, the clients (other than that they will all start running faster!).

In addition to this improved flexibility in your applications, SOAs are also designed to promote better reuse. Once you've started to provide a set of services for your business, creating new applications or business processes becomes much easier. Simply build on what you have while adding any additional services as you go. This allows organizations to move away from making big version changes to an entire application and instead promote a series of continuous, small changes to the different services, processes, and clients in their environments. This adds a significantly higher level of maintainability to any application.

Finally, let's not forget performance, scalability, and manageability. Companies like Oracle, IBM, and Intel are moving toward grid computing solutions. "The Grid," as it is known at Oracle, is a runtime architecture that's designed to intelligently manage all tiers of all your distributed applications from one unified location. The grid ensures that the right applications get the right resources and computing power at the right time. And while any J2EE application can run on the grid, companies can really increase the benefits they get from the grid by having a supply of reusable business services that are deployed and managed using a variety of different grid policies. As you might have guessed, this is a perfect job for an SOA.

Take another look at service-oriented architectures and decide which are the best service-oriented interfaces for your requirements. By implementing an SOA, companies can gain many benefits over traditional application architectures, regardless of whether or not they have committed to Web services technologies. Developers can start utilizing SOAs today and get increased flexibility and better control over their applications, while at the same time aligning themselves with technologies such as Web services and grid computing to gain added advantages in the future.

99.9% of the world won't find these screen shots terribly exciting. But if you're in the other 0.1%, yeehaw.

Use Crystal Reports 10 with your J2EE applications

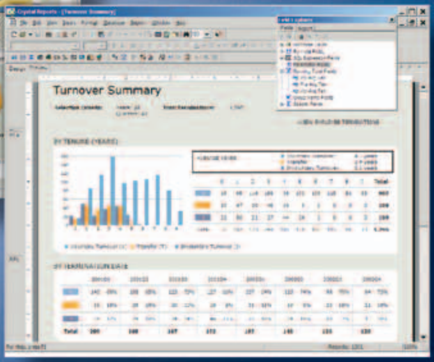
New 100% Java reporting component. Deploy reports across Unix, Linux and Windows

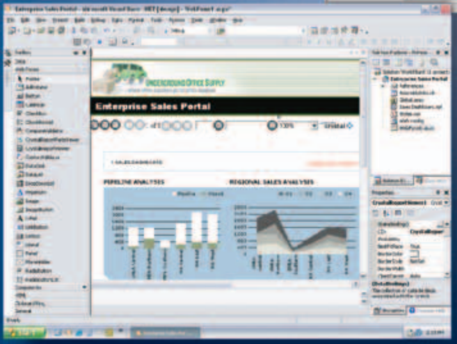
Extend Crystal Reports with Crystal Enterprise. Get world-class web report publishing, management, and scalability

Visual Designer simplifies data connectivity

Deliver diverse data formatting options within your presentation layer

Access data natively, or via ODBC, JDBC and OLE DB





Expedite .NET and J2EE report integration

Design and integrate reports from within popular IDEs

Flexible Java, .NET and COM SDKs support the tight integration of report interactivity including: group tree navigation, exporting, printing, and drill down

New Crystal Reports 10.

The best in business intelligence now offers the best in business reporting. New Crystal Reports® 10 is a faster and simpler way for developers to integrate dynamic data into applications and implement high-quality viewing, printing, and exporting. Learn more about Crystal Reports 10 and Crystal Enterprise™ 10, and access technical and evaluation resources at www.businessobjects.com/v10/047 or contact us directly at 1-800-877-2340.

BUSINESS OBJECTS®

Using JAXB in Enterprise J2EE Applications

Incorporate XML data and processing functions *Part 1*

by Tilak Mitra

Tilak Mitra is a certified IT architect at IBM. He specializes in mid-to-large-range enterprise and application architectures based on J2EE, MQ, and other EAI technologies.

tmitra@us.ibm.com

It has been well proven over the past few years that the best form of information exchange (in a typical B2B and B2C environment) is through XML. There are various XML-based standards (schema) for both the horizontal and vertical market sectors and there are ongoing efforts to move toward a standardized format in the various industry sectors.

With the proliferation of an XML-based information exchange, the industry is bound to write lots of Java code to consume XML Schema–based documents. Java Architecture for XML Binding (JAXB) provides a convenient way to bind an XML Schema to a representation in Java code, making it easy for developers to incorporate XML data and processing functions in applications based on Java technology without having to know much about the details of XML parsing.

How It Works

The use of JAXB starts from an XML Schema. Typically in an enterprise application, an XML Schema is defined that constitutes the business domain objects and their interrelationships.

The JAXB (binding) compiler creates a set of classes and interfaces from the XML Schema (see Figure 1). These sets of classes and interfaces are referenced and used in the application. The application developer has a rich set of JAXB APIs that he or she uses to convert a Java object tree/structure (made up of the instances of classes generated by the binding compiler) to an XML document (that conforms to the XML Schema). The process of converting an XML document into a Java object tree is as seamless and easy as the former and the beauty of it all is that the developer does not have to write a single line of XML parsing routines in either of the conversion processes.

The process of converting a Java Object Tree to an XML document is known as marshalling, whereas the reverse process of converting an XML document to a Java Object Tree is called unmarshalling.

The process of creating the classes and interfaces from the XML Schema utilizes the JAXB binding compiler (xjc.bat or xjc.sh) that's included with the installation (see Resources section).

```
xjc.bat -p <package name> -d <working
directory>
```

The –p option specifies the Java package for the generated classes and interfaces while the –d option specifies the working directory.

Once the classes and interfaces are generated, they can be used in the enterprise application. There are two typical usage scenarios.

1. ***Unmarshal an XML document to a Java object tree***

- To achieve this, an instance of a JAXBContext object needs to be created:

```
JAXBContext jContext =
JAXBContext.newInstance("<package name>") ;
```

where package name contains the JAXB generated classes.

- An unmarshaller instance is created:

```
Unmarshaller unmarshaller =
jContext.createUnmarshaller() ;
```

- The XML document is read in and a handle to the root Java object in the XML document is obtained:

```
 Library library =
(Library)unmarshaller.unmarshal(new
FileInputStream("library.xml") ;
```

library.xml is an example XML document that conforms to the schema file from which the JAXB-generated Java classes and interfaces are created, and Library is the root object in the XML document.

Once a handle to the Library object instance is obtained, we're in the Java universe! The developer can use the power of Java to traverse through the object tree and use the same in the application, as required.

2. ***Marshal a Java object tree to an XML document***

A JAXB implementation–provided class called ObjectFactory is used to cre-

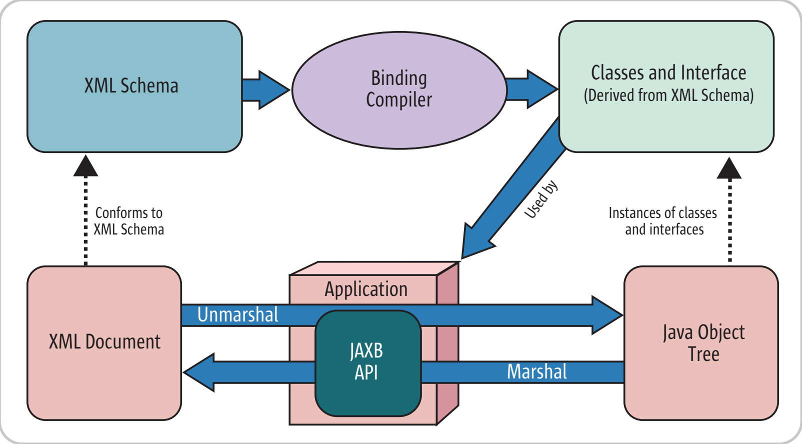


Figure 1 JAXB architecture

APRIL
BUZZ

Jtest®

Automate unit test case generation for JUnit and Java with Parasoft Jtest®

Verifying Algorithms Through Unit Testing

Complicated algorithms are typically contained within one class or function so that developers can easily manage them. How then, can you test the functionality of your algorithms and verify that they correctly solve problems before you have a full application written? The answer lies in unit testing.

Unit testing is the perfect strategy for verifying and testing algorithms. Unit testing involves testing software code at its smallest functional point, which is typically a single class. Each individual class should be tested in isolation before it is tested with other units or as part of a module or application. By testing every unit individually, most of the errors that might be introduced into the code over the course of a project can be detected or prevented entirely.

I suggest that you investigate the benefits of performing unit testing to verify the functionality of your algorithms. Performing thorough unit testing reduces the amount of work you will need do at the application level, and drastically reduces the potential for errors.

By the way, unit testing is a fundamental part of Parasoft's Automated Error Prevention (AEP) methodology. You can find more information about Automated Error Prevention at **www.parasoft.com**

*– **Adam Kolawa**, Ph.D.*
Chairman/CEO of Parasoft

Introducing Parasoft® Jtest®

Parasoft Jtest is the first and only automated unit testing tool for Java development.

With just a click, Jtest reads and analyzes code – quickly creating harnesses, stubs and test inputs – and tests without user intervention. Jtest also enables you to automate regression testing and coding standard analysis. For loyal JUnit users, Jtest is designed to fully support existing test cases and automate the creation of new JUnit-style test cases.

Learn how Jtest can enhance JUnit capabilities...

Download a free eval copy of Jtest along with our informative new white paper entitled "Automated Java Unit Testing, Coding Standard Compliance, and Team-Wide Error Prevention."

For Downloads go to **www.parasoft.com/jdj_04**

Or call 888-305-0041

PARASOFT®
We make software work.™

FEATURES

- Fully integrated with JUnit and Eclipse
- Finds and fixes errors fast
- Automatically generates JUnit test cases
- Customizable testing and reporting

BENEFITS

- Makes error prevention feasible and painless, which brings tremendous quality improvements, cost savings, and productivity increases
- Makes unit testing and coding standard compliance feasible and painless
- Encourages the team to collaborate on error prevention

PLATFORMS

Windows 2000/XP
Linux
Solaris

Contact Info:

Parasoft Corporation
101 E. Huntington Dr., 2nd Flr.,
Monrovia, CA 91016
www.parasoft.com

ate instances of the classes and interfaces that are generated by the JAXB binding compiler (during the generation process). This class uses the Factory pattern to create instances of the generated classes and this is the only way the class instances may be created.

Consider a small example in which a Library contains a list of Books in which each book has a title and a price field. The steps to create an XML document from the Java object tree are as follows:

- Obtain an instance of ObjectFactory:

```
ObjectFactory factory = new ObjectFactory()
;
```

- Create class instances:

```
Library library = factory.createLibrary() ;
Book  bookOne = factory.createBook() ;
bookOne.setTitle(“Design Patterns”) ;
bookOne.setPrice(“50.00”) ;
Book bookTwo = factory.createBook() ;
bookTwo.setTitle(“Analysis Patterns”) ;
bookTwo.setPrice(“45.00”) ;
library.add(bookOne) ;
library.add(bookTwo);
```

- Create an instance of the Marshaller object (from JAXBContext object instance as in scenario 1):

```
Marshaller marshaller =
jContext.createMarshaller() ;
```

- Marshal the Java object tree to an XML document:

```
marshaller.marshal(library, new
FileOutputStream(“library.xml”)) ;
```

JAXB Customizations

The power and flexibility of JAXB is further augmented by its customization feature that's added on top of the schema bindings. To add specific functionality to an application, JAXB binding customizations are used. These customizations are read and interpreted by the JAXB compiler. Customization is affected by annotating a schema with binding declarations that either override or extend the default bindings.

Customization has four scopes:

- ***Global:*** A customization defined with the <globalBindings> element has a global scope applicable to all schema elements and all other schemas that are imported into the schema in which this customization is defined.
- ***Schema:*** A customization defined with a <schemaBindings> element has a schema scope and is only applicable to the schema in which the customization is defined.
- ***Definition:*** A customization value in binding declarations of a type definition and global declaration has definition scope, which covers all schema elements that reference the type definition or the global declaration.
- ***Component:*** A customization value in a binding declaration has component scope if the customization value applies only to the schema element that was annotated with the binding declaration.

Although a detailed discussion about customization is beyond the scope of this article, it's worthwhile mentioning a few customization artifacts that are used more frequently in a typical JAXB usage scenario.

Customization bindings can be made at a global level of declaration that applies to all the defined elements in the XML Schema. Listing 1 provides a typical global customization binding.

Notice how in the listing defining the collectionType attribute tells the compiler that the type of collection used in the generated classes is of type ArrayList. The <xjc: serializable> element ensures that all the generated classes implement the Java Serializable marker interface. The <jxb: package> element's name attribute denotes the Java package in which the generated classes and interfaces are created.

All these elements' attributes can be tuned and configured to suit the requirements of the application.

Customization of the default binding can also be made at the element and its attribute's level. For example, an element can be adorned with its Javadoc by using annotations. Custom property names can also be specified that when defined, generate getter and setter methods for the property. This is illustrated in Listing 2. (Listing 2 can be downloaded from www.sys-con.com/java/sourcec.cfm.)

Notice how in the listing the <jxb: javadoc> element is used to create the documentation for the generated class. The <jxb:property> element is used to name an instance variable inside the generated class (Library class in this case) and generate its getter and setter methods. The generated class hence will have two methods: getBookList() and setBookList(…).

A detailed explanation of these bindings can be found in the Resources section (JAXB User's Guide).

Thinking in Terms of JAXB

In a typical enterprise application that's comprised of various application tiers, data is exchanged between these tiers to realize the business functionalities. It's a good design principle to create a data object model that can be used for inter-tier data exchange. This data object model is a good candidate to be represented in an XML Schema with JAXB customizations. With this design in place, data can be exchanged in either XML format or as Java objects while the JAXB libraries can be used to convert between XML and Java in a seamless fashion.

Summary

This article introduced the basic concepts of JAXB, how it works, and how it can be used in a Java-based enterprise application. It also provided a sneak peek at how JAXB customizations can be used to tailor an XML Schema to conform to the application requirements.

Part 2 will take a concrete example of an XML Schema and discuss the process of generating the classes and interfaces from a valid XML document and also the reverse process of creating an XML document from a Java object tree. Stay tuned!

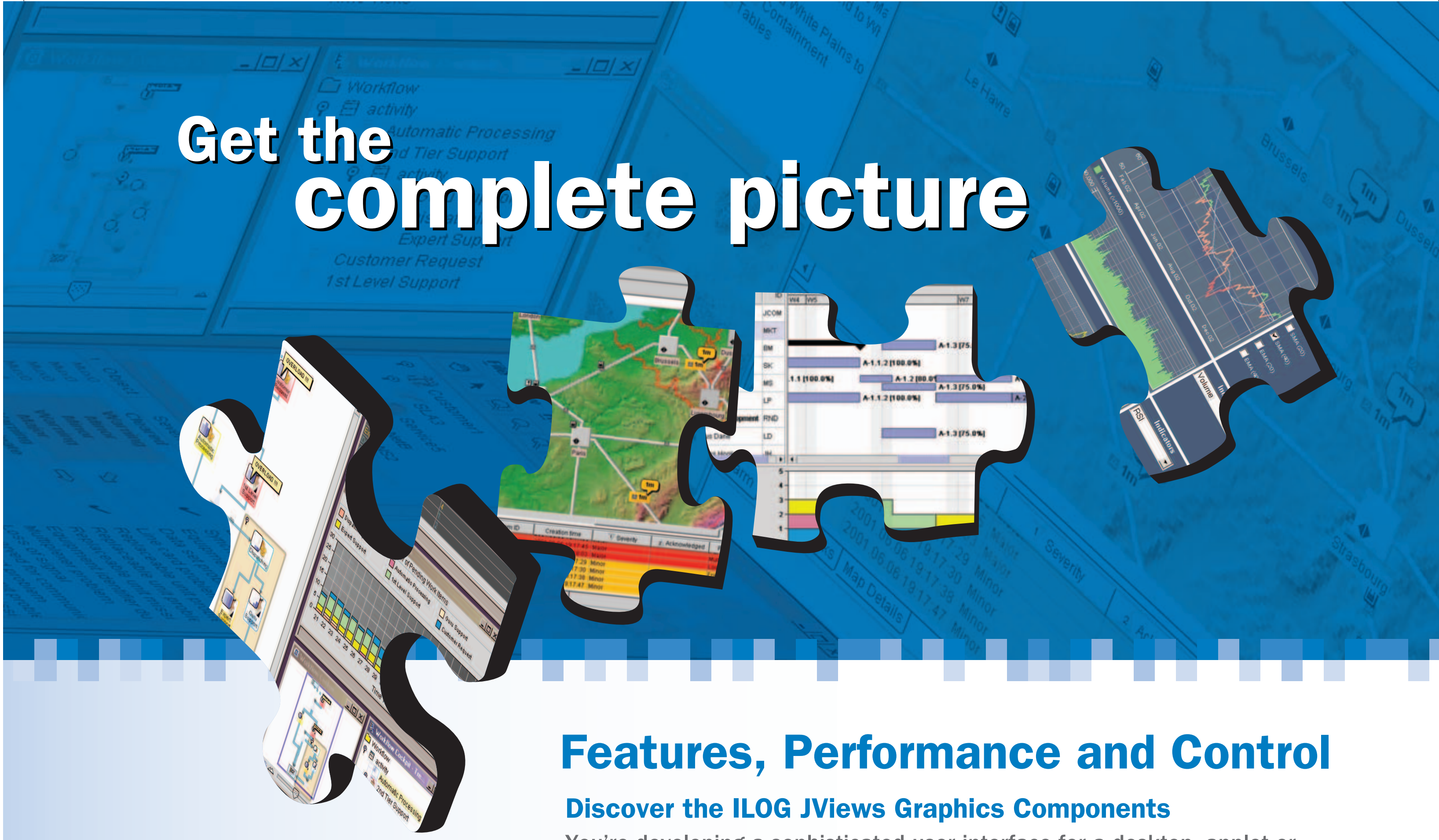
Resources

- *JAXB User's Guide:* http://java.sun.com/xml/jaxb/users-guide/jaxb-using.html
- *JAXB Specification:* http://java.sun.com/xml/downloads/ jaxb.html
- *JAXB API Specification:* http://java.sun.com/webservices/ docs/1.3/api/index.html
- *JAXB Reference Implementation:* http://java.sun.com/webservices/downloads/webservicespack.html

(*Note:* The Reference Implementation of JAXB comes packaged inside the Java Web Services Developer's Pack [JWSDP]. Once this is installed, the JAXB compile time and runtime libraries are available in the <install-root>\jaxb directory.)

Listing 1

```
<xsd:annotation>
 <xsd:appinfo>
<jxb:globalBindings
collectionType="java.util.ArrayList"

fixedAttributeAsConstantProperty="true"
        generateIsSetMethod="false"
        enableFailFastCheck="false"
        choiceContentProperty="false"

underscoreBinding="asWordSeparator"
        typesafeEnumBase="xsd:NCName"

typesafeEnumMemberName="generateError"

enableJavaNamingConventions="true"
        bindingStyle="elementBinding">

    <xjc:serializable />

</jxb:globalBindings>
<jxb:schemaBindings>
 <jxb:package name="com.ibm.domainob-
jects" />
</jxb:schemaBindings>

</xsd:appinfo>
</xsd:annotation>
```



Features, Performance and Control

Discover the ILOG JViews Graphics Components

You're developing a sophisticated user interface for a desktop, applet or servlet application – it needs to provide displays that go far beyond what Swing and HTML offer. How can you be sure it will have the features, performance, customization and scalability to enable your end-users to make better more informed decisions, faster?

With ILOG JViews, you get comprehensive graphical libraries & tools, resources, and maintenance services so you can focus on the implementation, confidently completing your application in less time and at less cost.

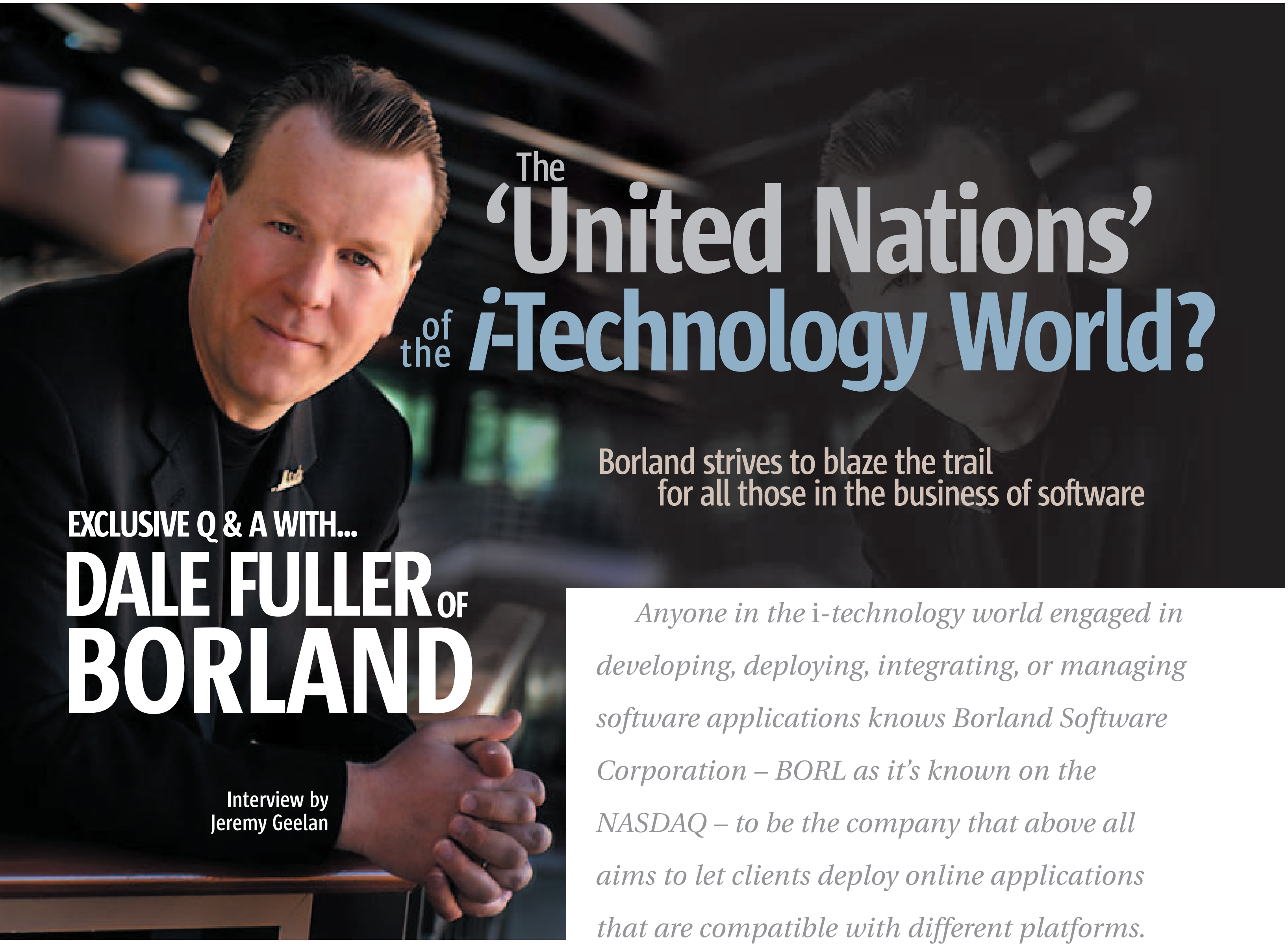
Quickly and easily build:

- Gantt and resource displays
- Graph layouts, diagrams, workflows
- Geographic map displays
- Realtime data charts
- Custom monitoring and control screens
- Network and equipment management screens

Get a JViews Info Kit – Learn more, test drive an Eval.
Go to: jviews-info-kit.ilog.com or **Call: 1-800-for-ILOG**

ILOG®

Changing the rules of business™



The 'United Nations' of the *i*-Technology World?

Borland strives to blaze the trail for all those in the business of software

EXCLUSIVE Q & A WITH...

DALE FULLER OF BORLAND

Interview by
Jeremy Geelan

Anyone in the i-technology world engaged in developing, deploying, integrating, or managing software applications knows Borland Software Corporation – BORL as it's known on the NASDAQ – to be the company that above all aims to let clients deploy online applications that are compatible with different platforms.

This gives Borland's president and CEO, Dale Fuller, a unique vantage point from which to comment not just on Java or .NET, but on all manner of current technologies. Fuller also recognizes, along with *JDJ*, that these days "Everyone from the CIO through to the developer is in the business of software."

Accordingly, this month's *JDJ* "Question & Answer" session gave Fuller an opportunity, from his corporate world headquarters in Scotts Valley, CA, to deal with a range of issues from ASPs and Web services to SOA, Linux, application life-cycle management, "invisible middleware," and Borland's future role as IT's nearest equivalent to the United Nations – serving all those in the world business of software, no matter what particular brand they may owe allegiance to.

JDJ: We have the basic building blocks and standards in place for Web services, and people are using them. Where do you think we will go from here?

Dale Fuller: We'll need tools that will allow us to put the Web services standards and building blocks to use in a higher-level way. Web services lend themselves to a more business process–oriented approach toward applications and we are going to require development tools that work in this more process-oriented way. This means tools that can visualize the solution at a higher level – think UML models, but UML models that serve as more than just a blueprint, and instead become the basis for the application itself. The software industry must go beyond SDKs and toolkits and toward a higher-level way of building and integrating applications.

JDJ: Whatever happened to confuse the meaning of Web services? Do you think that it's because Microsoft focused at the beginning on Web services from the consumer, individual point of view rather than from the back-end IT point of view?

DF: Looking at the origins of the Web services story, you'll see that the "marketing" side didn't embrace the terminology that IT and developers were accustomed to in distributed computing.

The terms that we all understood at the time were things like remote invocation, request brokers, distributed objects, call backs, functions, and APIs. When Web services was introduced, there was a significant emphasis on the "services" aspect of this new technology opportunity. This made Web services appear to be something more magical and dot.com oriented, rather than a simple and practical technology for building networked and distributed applications.

This probably had to do with the timing of the emergence of the technology; remember at the time the SOAP specification emerged, the hot prospect of the day was application service providers (ASPs). I think many in the industry saw a natural fit and wanted Web services to ride the coattails of that wave.

JDJ: Does Borland see a role for Linux within Web services?

DF: Wherever there is a Web server there is a role to host Web services, and there are certainly a lot of Apache Web servers running on Linux.

JDJ: Salesforce.com's CEO Marc Benioff prefers not to use the term "Web services" at all but refers to the distributed application architectures that we've all been pursuing for years as "client/service." Do you agree? Should we be talking about "client/service" architectures now?

DF: To the extent that "Web services" describes just one of the possible benefits of the technology, the title can be seen as a misnomer. It is certainly a good idea to talk about the other benefits of Web services such as "client/service," "peer to peer," and "business to business." However, over the last five years the term Web services has become pretty entrenched in IT vocabulary for a simple and fundamental idea, the practical applications of which are well beyond a single architecture.

JDJ: How do we straighten things out so that from here on technology buyers can understand it all?

DF: If the software industry were to talk about Web services as the simple messaging protocol it was initially designed to be, more organizations would begin to see the potential benefits for using Web services in their businesses. This could very well be a case where actually talking about technology could simplify the mystery of the higher-level marketing that has dominated the technology since its inception.

JDJ: What about the buzzword of 2004, SOAs – do you agree with those who say that enterprise adoption of service-oriented architecture will take many years? If so, then in the meantime what should programmers be doing?

DF: SOA is happening as we speak and IT organizations are quickly realizing that the business advantages of SOA are simply too large to be ignored. The need to quickly respond to changing market and organization conditions and to maximize business agility is the key drive behind service architectures. Quality, reliability, and maintainability are main concerns for CIOs today as they look at their software systems. However, shifting to SOA requires greater architectural discipline across the entire organization and throughout the application life cycle.

To be successful at it, IT groups cannot work in isolation; strong team collaboration is required to define, implement, and manage SOA. SOA also incurs greater performance overhead, a risk for systems that have strictly enforced response times. This requires closer performance monitoring and a secure, scalable, distributed computing environment.

> "Everyone from the CIO through to the developer is in the business of software"

JDJ: We heard reliable rumors at *JDJ* when "Tiger" was unleashed earlier this year that it went through unanimously, but that three of the heavyweight companies among the 14 members on the JSR committee concerned were complaining that there's not enough CORBA support. Does Borland have a position on CORBA support in Java?

DF: The CORBA ORB capabilities offered within J2SE provide basic support for such technologies as the Internet Inter-ORB Protocol (IIOP), the communication foundation with CORBA. While J2EE uses Remote Method Invocations as its communication foundation, the EJB specifications within J2EE define that it is IIOP that must be used in order to access EJBs.

This hasn't changed even with the latest J2SE 1.5 release. As organizations implement CORBA servers throughout their enterprise, they must rely on enterprise-grade CORBA ORBs that provide extensive CORBA services beyond those offered within J2SE/J2EE. Some of these services include security, transactions, notification, and messaging as well as support for other enterprise grade qualities of service such as failover, load balancing, and clustering.

JDJ: Still on Java, what's your position on the JCP – is it the right way of doing things?

DF: Borland is committed to the Java Community Process and to supporting Java standards. We believe that the JCP was the right model for a specific time in the evolution of Java technology. Sun maintained "executive" control as a lever to keep the technology evolution moving quickly. However, as Java technology matures, it could be time for Sun and the JCP to look at an increasingly participative, democratic model that is more reflective of open standards.

JDJ: Would/could anyone else safely be the custodian of Java? The open source community, for example?

DF: We may have reached a stage in the evolution of Java technology where it is much less vital to have a "custodian" as we have had in the past.

Where does Borland stand in the *i*-technology world?

DF: Borland has solutions available today that provide management, control, and visibility to implement SOA across the entire organization. Our application life-cycle management strategy is designed to reduce risk and uncertainty, helping to ensure that developers are not working in isolation, but are quickly and reliably building solutions in line with business needs. Our goal is to make standards easier to use while avoiding vendor lock-in. And that’s what’s made us #1 in Java development.

JBuilder has always found ways to make Java and J2EE easier to use. We continue to do that with JBuilder X. JBuilder X offers a visual struts designer that takes advanced Web development in Java to the next level of productivity. As new Java standards emerge, we will continue to make them easier to use, and we’ve taken a leadership role in extending that productivity throughout the entire application life cycle with solutions such as Borland Enterprise Studio for Java. This combines JBuilder development technology with our Together solution for modeling, Optimizeit for performance management, CaliberRM for requirements management, and StarTeam for change management, as well as Borland Enterprise Server and JDataStore for deployment.

And products such as Borland Enterprise Server; Borland Deployment Op-Center; and Janeva for .NET, J2EE, and CORBA interoperability were designed to enable architectural heterogeneity.

JDJ: While on the subject of open source, what overall effect on Java do you foresee from the compelling economics of Linux?

DF: Linux is driving the adoption of low cost, open source servers in the marketplace. Java is viewed as the de facto application platform for the building and deploying of applications on top of that operating system. You could see Linux driving increased demand for J2EE/JSP/servlet-based applications.

JDJ: Do you think J2EE is too complex? If so, what’s the best way forward?

DF: It isn’t a question of J2EE being too complex. J2EE is a layer between the complexity of network-based utilities, such as DB, directory, security, and transaction processing.

What’s necessary is the ability to provide the developer with a better sense of “context” as to where he or she is in the process of building new applications that plug into an existing infrastructure or in the process of deconstructing and rebuilding existing applications.

> “Quality, reliability, and maintainability are main concerns for CIOs today as they look at their software systems.
> However, shifting to SOA requires greater architectural discipline across the entire organization and throughout the application life cycle”

JDJ: Why do you think Sun’s Jonathan Schwartz said to *JDJ* in February: “Middleware is history”? Is middleware in fact just beginning? Or is Schwartz right in saying that end-to-end “systems” will supplant it?

DF: Borland has always had a strong relationship with Sun, but on this point, we’ll have to lean toward the IBM billboards that dot Highway 101 in northern California. Their latest slogan is: “Middleware is everywhere. Can you see it?”

Sun provides an integrated “Java Enterprise System” that offers all of the basic components of shared services, from directory and identity management to Web services, e-mail, and clustering. While Borland does offer an application life-cycle management solution for Java, we prefer to see ourselves as the Switzerland of software development – providing our customers with the freedom to choose from a variety of vendors’ technologies depending on what works best in their organization’s IT environment.

JDJ: As a major business leader, what have you found to be the most compelling aspect of the technology space? And what’s the least attractive aspect?

DF: Most compelling: the fact that who we define as a software person today is drastically changing to recognize that everyone from the CIO through to the developer is in the business of software. The fact that the pace of change and innovation is forcing globalization and making the world smaller by bridging geographic and cultural divides. The fact that companies like Borland who helped drive the PC evolution are still making software pervasive in all that we do today – such as the use of our technologies in projects as broad as St. Judes Children’s Hospital and the NASA Mars Rover Mission.

Least compelling: when a vendor’s self interest gets in the way of freedom and innovation.

JDJ: After such great success with the notion that Borland is “the developer’s Switzerland,” will there ever be a need for you to choose – once and for all – between J2EE and .NET, for example?

DF: The trend is definitely toward greater heterogeneity. The analyst groups predict that moving forward it will be a two-horse race between J2EE and .NET. Borland would have to agree, because at the end of the day, it is about having the right technology for the problem at hand. Borland exists to ensure that our customers have that freedom of choice.

JDJ: If ever you abandoned the “Swiss” metaphor, what other country could Borland usefully emulate and invoke to characterize its unique role?

DF: I would have to say that we wouldn’t be a country at all – we’d have to be seen more like the United Nations. We don’t make the laws governing software development, but we do set out to help resolve conflicts between programming languages and break down the barriers between people and technology across the software development paradigm. We see ourselves as the end user’s champion – one of the last bastions of independence in the industry, playing a key role in helping to formulate the policies and standards that affect the industry.

Scalability of J2EE Applications

Effective caching

by Stefan Piesche

Sooner or later all architects and developers of large-scale J2EE products face the same problem: their software's response time gets slower and slower, and the scalability of their solution is ending. This article investigates caching solutions that promise to help; sheds some light on their limitations; and describes an easy, lightweight, and effective caching mechanism that solves most of the issues.

Note: This article does not assess all possible ways of caching nor does it take solutions such as commercial external caching products into account.

The Problem

Whenever we build distributed software for a large scale – whether it's J2EE or not – we face the same challenge: keep response or transaction times low while increasing user load. The main problem is that essentially all software systems scale exponentially at some point (see Figure 1). Architecting and implementing a solution that keeps scalability linear and leaves enough room for increasing load as the business grows is a difficult task that requires experience.

A good architect keeps traffic, transaction times and volume, persistence layer design, and caching in mind when he or she drafts the first layout of a new architecture. Understanding concurrent access by *n* users on *m* data items is one of the major things an architect looks for.

Stefan Piesche is a principal architect for the Cobalt Group (HQ in Seattle) responsible for large-scale, distributed systems based on J2EE. In the past years he's worked on several large-scale systems in Europe in the financial and airline industries.

spiesche@austin.rr.com

Possible Solutions

Minimizing traffic in all tiers is the primary objective when creating a scalable solution. Figure 2 shows a typical three-tier system.

While the persistence tier in modern databases already provides significant caching capabilities, it's rarely enough for large-scale systems. What do other mechanisms do to increase performance and scalability, and to what tier/layer do they apply?

Stored Procedures?

I mention this because aside from caching, one suggestion I always hear is using stored procedures. I'd like to encourage everyone to consider different options. Using stored procedures splits the persistence layer over two physical tiers and usually improves only single user performance.

If you look at your application server's console, you might see, for example, that of the 500ms a servlet or JSP request takes, only 100ms are spent on the DB transaction side. Squeezing another 30ms out by using stored procedures rarely makes your system scale – you still need DB connections, cursors, and other resources.

Persistence Layer Caching

The easiest way to cache in J2EE systems is with entity beans (if we say entity beans let's only talk about CMP for the moment); I can hear the readers moan, but the fact remains: they are the only "good" way of caching in J2EE solutions. Why? Because the maximum cache size is controllable by setting the maximum number of beans and because the resource is in control of the container, as they can be passivated if memory is short. Usually, they are the only resource that is clusterable as well.

Why would most developers and architects say entity beans are bad for your performance? Because they are. In a single request use case, they have significant overhead compared to direct JDBC. But even in scalability assessments, entity beans often come out last, because their usage as a cache is determined by the possible cache hit rate, just like any other cache. The cache hit rate is determined by the number of reads versus the number of data items versus the number of writes.

Ultimately, if you use entity beans you really need to know what you're doing. While that might be true for any out-of-the-box mechanisms a

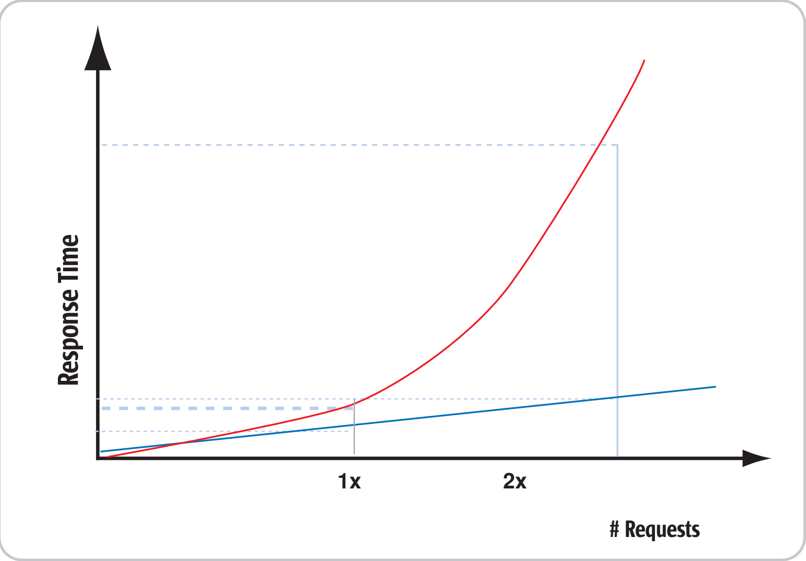


Figure 1 Linear versus exponential scalability

Apollo 17 Scientist-astronaut retrieving lunar samples. December 1972

ExtenXLS, the right tool for complex reporting™

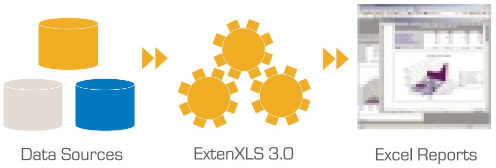
ExtenXLS 3, the 100% Java reporting toolkit

When HP, Ford, and Qwest Communications selected a solution for their mission critical reporting projects, they turned to ExtenXLS. With unparalleled support for complex Excel files, ExtenXLS enables creation of feature-rich Excel reports with charts, formatting, text, macros, formulas, and numerical data from any data source. Combining the flexibility of Java with the functionality and ubiquity of Excel, take a giant leap forward in reporting capabilities without abandoning existing business logic.

Best-in-class Excel compatibility, robust features, and a company dedicated to its products. ExtenXLS, the right tool for complex reporting.

Download a free evaluation copy today: **extentech.com/jdjxls/**

EXTENXLS³
JAVA | XLS REPORTING TOOLKIT™



- ✓ Fully supported
- ✓ Commercial-grade Excel compatibility
- ✓ Streamlined, easy to use API
- ✓ In-memory Formula Calculation
- ✓ 130+ Formula Functions Supported
- ✓ Best-in-class 3D Named Range support
- ✓ Add hyperlinks to Cells for drill-down reports
- ✓ Serialize & copy Sheets between WorkBooks
- ✓ Total control of fonts and cell formatting
- ✓ Comprehensive sample code and templates
- ✓ Used worldwide by global 1000 companies

extenTECH™

www.extentech.com

Image AS17-145-22157 provided courtesy of NASA. NASA does not necessarily endorse Extentech, products or service, or companies, trademarks, products or services herein mentioned.
Java is a Registered Trademark of Sun Microsystems Inc. Excel is a Registered Trademark of Microsoft Corporation. All other trademarks mentioned herein are the property of their respective owners.

container provides, it's especially true for entity beans. It's easy to get it wrong and a lot of containers have less than mediocre support for entity beans.

Entity beans make sense if:

- Your reads and writes are few, then scalability is not your concern anyway and CMP EJBs are just as convenient.
- Your reads are many, your writes and number of data items are few – this means maximum cache hit rate – you have just a few items to cache (most containers only perform well with a few thousand entity bean instances per CPU), and it rarely becomes stale because you hardly write.

In all other cases, entity beans just make things worse due to their management overhead. Figure 3 shows that cache efficiencies (like entity beans) depend on the number of reads versus the number of writes versus the number of rows (which is an oversimplified perception and not real math). Caching with entity beans works well within the green area.

One important fact needs to be considered as well: some application servers (WLS 6, WebSphere) do not support EntityBean clustering/caching in clustered infrastructures. In other words, they often support only the caching of read-only entity beans if you run a cluster, which rules straight CMP out completely to increase scalability.

Let's have a quick look at BMP (mainly read-only or read-mostly BMP). These type of entity beans can be used to solve the problem of too many entity bean instances by allowing you to change the caching granularity: while CMP caches on a per-data-row basis, RO BMPs can essentially cache on any desired granularity level and are basically similar to the caching mechanism I'll discuss later. However, they still have a few disadvantages, such as the entity bean management overhead or (depending on your container implementation) the fact that they usually are – like all entity beans – single threaded: only one request at a time can access the cache.

In all other cases (mixed reads/writes, lots of data, few reads many writes, etc.), how do we make our software scalable?

Web Tier Caching Using HTTP Session

If persistence layer caching through entity beans is ruled out, we have two tiers left where we could cache.

The most obvious choice developers often make is HTTP session caching. Since it caches at the uppermost tier, it should be most effective at minimizing traffic, right? However, using the HTTP session as a cache makes architects of large-scale systems shudder.

First, it caches on a per-session basis: it helps if one user performs the same or similar action 5,000 times but not if 5,000 users perform one action.

Second, the cache invalidation and GC is based on the session time-out – usually something like 60 minutes. Even if a user works for 10 minutes in your system, the data is cached for 60 minutes, which makes the cache size six times as big as it needs to be, unless you invalidate your session manually.

Finally, it removes one important task from the container: resource management. Since this cache cannot be cleared by the container, it often causes problems since the container cannot GC these objects even if memory resources become short. The container's GC cycles become more frequent and the GC has to walk over a large set of mainly stale objects in your session, making the cycles longer than they need to be.

Singletons and Application Context

The last place to cache is in the business layer (the following mechanism could be used in the Web tier as well). Since the HTTP session is not very effective at caching in high-traffic systems, the next best choice is using singletons to cache objects or data from the database.

Singletons (just like the application context) have the advantage that they again cache for all requests, but still are not a container-managed resource. Frequently singleton caches are implemented as a plain Hashtable and are unlimited in size, which causes almost the same problems as HTTP session caching.

I'd like to recall a simple but effective caching strategy that is singleton-based and uses a container such as a mechanism of resource management to keep resource usage to a minimum.

LRU Caching

The strategy used is called LRU (least recently used), also known as MRU (most recently used). Essentially, it only caches objects that are used frequently by limiting the cache size to a fixed number of items (hence the name), just like a container pool size for EJBs, thus keeping resource utilization controlled.

How does this work? Essentially it's a stack: if an object is requested from the stack and it's not there (cache miss), it's inserted at the very top. If your cache size is 1,000 items and the cache is full, the last item will fall off the stack and effectively be removed from the cache (see Figure 4).

In case an object is on the cache, it will be removed and reinserted at the top (see Figure 5).

This way, the most often used items will remain at the top, and the least used

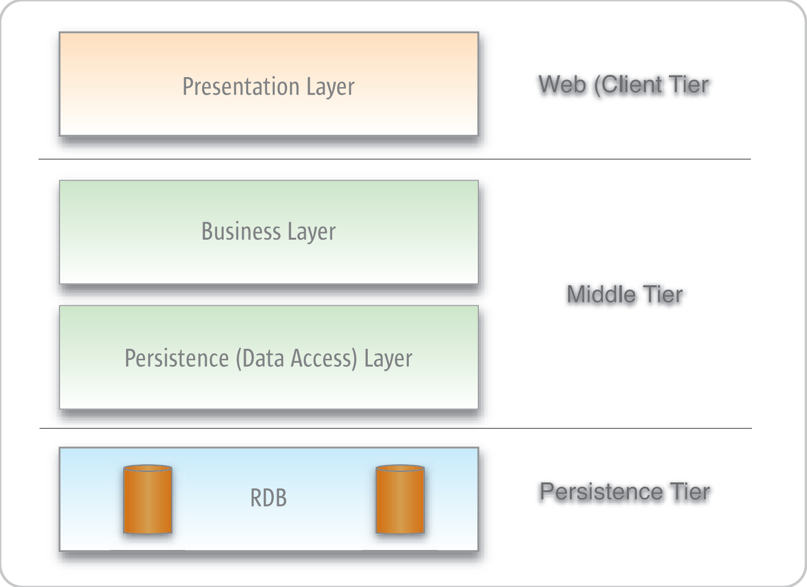
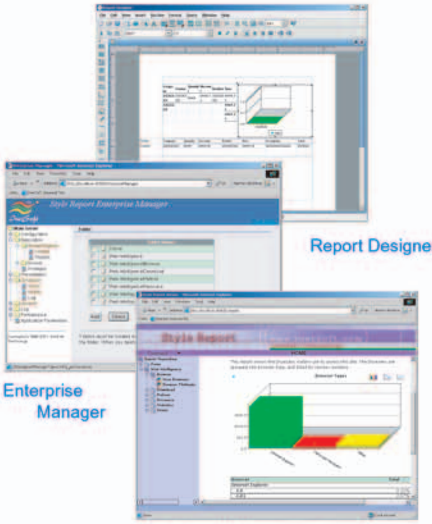


Figure 2 Three-tier system with business layer and persistence layer in middle tier

Java & Web Reporting:
with Java flexibility without Java complexity

Style Report:
Information Vantage Point



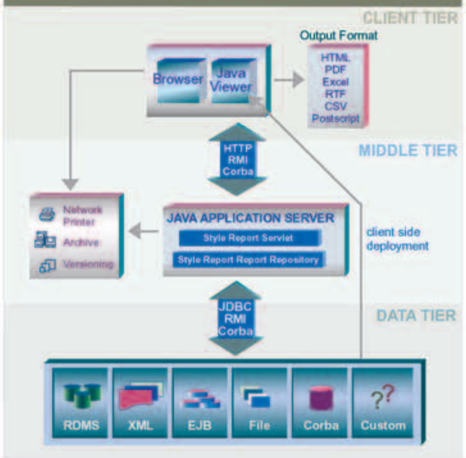
Report Designer

Enterprise Manager

Web Report Viewer/Composer

Style Report blends the best of Java and Web to provide an award winning Web enterprise reporting solution. Based 100% on open standards such as XML, JSP, Java and Web services, Style Report allows you fully leverage existing IT infrastructure and common technical skills. One Report, Multiple Channel delivery and zero client ad-hoc reporting gives end users the freedom and frees IT from time consuming repetitive work. Designed for integration, Style Report is deployed and managed as a standard Web application or Corba/RMI service requiring not special skills. Download a free evaluation copy now!

STYLE REPORT ARCHITECTURE DIAGRAM



Style Report Features	
Flexible Visual Designer with JavaScript or VB Script	☑
PDF, Excel, Word, CSV and other Export Format	☑
Database/XML and other Datasources	☑
Zero Client Browser Viewing	☑
Built-in Drilldown/Parameters	☑
Zero Client Ad-hoc Reports	☑
Virtual Private Security Model	☑
J2EE Drop-in Deployment	☑
Advanced Scheduling	☑
Report Versioning/Archive	☑
Demand Paging/Report Cache	☑
Portal/JSP integration	☑

JAVA DEVELOPER'S JOURNAL JDJ READERS' CHOICE AWARD

InetSoft
open standards innovation

www.inetsoft.com
JDJ special http://www.inetsoft.com/jdj

Phone: 888-216-2353 (USA)
Phone: 732-424-0400 (Intl)

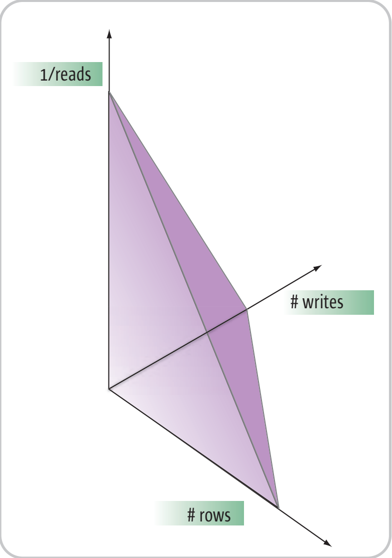


Figure 3 Entity beans cache efficiency diagram

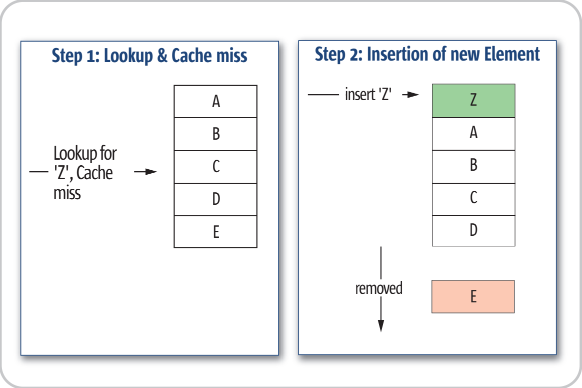

Step 1: Lookup & Cache miss
Lookup for 'Z', Cache miss
A
B
C
D
E
Step 2: Insertion of new Element
insert 'Z'
Z
A
B
C
D
removed
E

Figure 4 Cache miss and insertion

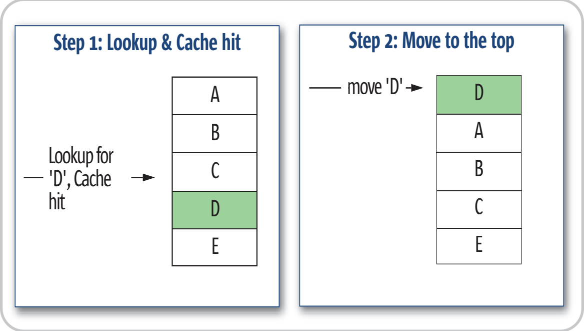


Figure 5 Cache hit and object moving to top

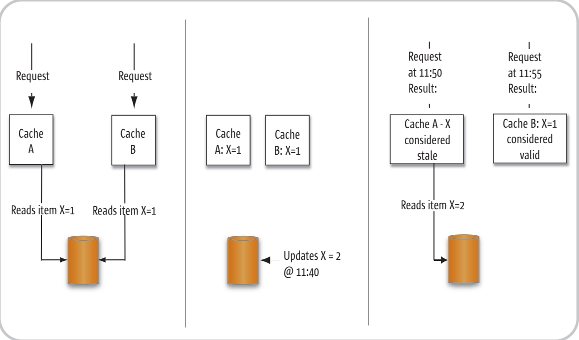


Figure 6 Caches getting out of sync

items will eventually drop off the stack. You can even keep track of your hits and misses easily and either query this information to reconfigure your cache or grow and shrink the maximum size dynamically. This way, you minimize usage of resources and maximize cache effectiveness. The stack implementation depends on your needs: choose an unsynchronized model if necessary to allow concurrent reads and minimize overhead.

Cache Invalidation

This cache works best in read-only or read-mostly scenarios. Unless you implement write-back or other write cache synchronization schemes or don't care that the cache is out of sync with the data source, you'll have to invalidate the cache, which decreases the cache hit rate and efficiency. For example, you can implement write-through caches fairly easily using dynamic proxy classes (JDK 1.3 introduced support for dynamic proxies) but that is a topic for another article.

Singleton-based LRU caching still has the typical problem of all singleton-based caches: a singleton is not a singleton in distributed systems (J2EE for that matter) but unique per classloader or server context (if you're lucky), and it's not kept in sync in clustered environments. There are, of course, mechanisms to implement the synchronization of distributed resources; some of them are difficult to implement or have scalability or availability issues; some work just fine. Distributed caching is not easy and if your requirements force you to go down this path, you might be well served choosing a commercial caching product.

The fact that you have several unsynchronized cache copies in clustered environments can be a big problem. The easy solution is using timed caches (just like read-only entity beans), which means that if a cached object is a certain age, it's considered stale and will be dropped from the cache. This is sufficient in most cases, but let's look at the following scenario.

Let's assume our invalidation time is 30 minutes (an object older than 30 minutes is considered stale). Cache A caches an object at 11:15, Cache B at 11:35. If the data item the cache is referring to is refreshed in the database at 11:40, Cache A will have the correct value at 11:45 when it expires but Cache B won't have it until 12:05 (see Figure 6). The problem now is that for 20 minutes you get different results – depending on which server you hit and on the use case this can be a big problem.

The solution for these cases is a timed cache that is refreshed at fixed points in time every *n* minutes, like at 12:00, 12:30, 1:00, etc. The advantage is that now all your caches are somewhat in sync (as in sync as the clocks on your servers are). The disadvantage is that the load on your servers increases quite a bit every time the caches are cleared, because they're cleared completely.

Which way you go depends on your business requirements; adjusting your refresh cycles largely depends on your data update frequency versus the cache hit rate you would like to achieve.

Of course, there are a variety of other ways to keep distributed copies of caches in sync, but these are not easy to implement and have a variety of side effects to consider.

Open Source and Commercial Caching Implementations

If your caching needs are more complex, or if you just don't want to "roll your own," you might want to give JSR 107 a look. This is the JCache JSR that specifies a distributed, resource-managed, Java-based cache. Even though little progress has been made to provide a production-ready implementation, there are several open source projects and products that are close to a JCache implementation and might provide what you need.

Commercial caching products should be considered if your caching requirements are complex (clustered environments, etc.). As mentioned earlier, distributed caching is not as easy at it seems and relying on an enterprise-class product often saves time and trouble.

Building a scalable solution often depends on making the right decisions in persistence mechanism and in caching. How, when, and where to cache is the trick; I hope this article helped you make the right decision.

References

- *JSR 104:* www.jcp.org/en/jsr/detail? id=107
- *JCS and JCache at Apache:* http://jakarta.apache.org/turbine/jcs/JCSandJCACHE.html

Dralasoft is the technology
of choice for business process
management (BPM)

Technology that Xerox®
chose to embed in DocuShare®

Technology that won
Best of Show at AIIM 2004 in
BPM/Workflow category

Technology that is enterprise ready

Technology
that *you* should consider

Dralasoft provides workflow
processing as a component.
With a best of breed
visual design tool.
Dralasoft Workflow™ is a
comprehensive BPM solution.
We speak your language –Java, XML,
SOAP, LDAP, J2EE.

Did you know that leading software vendors embed Dralasoft Workflow?

Can you afford not to put Dralasoft on the top of your list?

Learn more about who is using Dralasoft by
visiting www.dralasoft.com/customers

dralasoft
Workflow Automation Simplified

www.dralasoft.com

Dralasoft, Inc. 10955 Westmoor Drive, Suite 110 Westminster, CO 80021 Email: sales@dralasoft.com Phone: (303) 468-6754 Fax: (303) 468-6758 eFax: (303) 479-9836

Evolving the JRE

...from a managed runtime to a manageable runtime

by Stuart Lawrence
and Bob Griswold

When Java was first released, it was immediately attractive due to its ease-of-use and the promise of WORA (write once, run anywhere). As it evolved, the value of the JRE abstraction has manifested itself in many ways not immediately apparent from the days of animated applets. For example, the widespread adoption of Java on the server helped to drive the development of several performance profiling and application monitoring tools and techniques. These tools and techniques bring great value to the Java platform but some also have significant limitations and drawbacks. This article surveys current tools and techniques and looks at new initiatives to evolve the JRE into a truly manageable runtime.

Managed Runtime Environments

One of the original key goals of Java was to provide a layer of abstraction that allowed a programmer to write code that was portable to many different hardware/OS platforms. However, the Java Virtual Machine (JVM) doesn't just offer programmers a machine-independent way to run their code, it dynamically *manages* executing code. The JRE was the first mainstream Managed Runtime Environment (MRTE) with characteristic properties such as:

- ***Bytecode:*** A machine-independent executable representation of application code

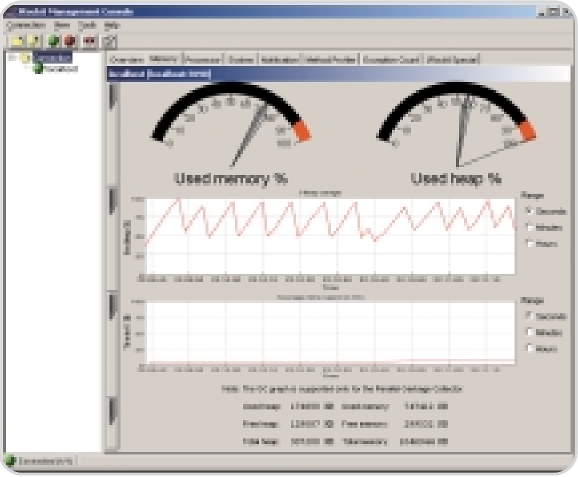


Figure 1 The JRockit Management Console

- ***Automatic memory management:*** Object allocation, garbage collection, dynamic heap resizing, and compaction
- ***Program security and correctness:*** Bytecode verification, sandbox for applets, no pointers, type checking, array bounds checks, and null pointer checks
- ***Dynamic optimization:*** The VM can adapt to the characteristics of the running application and selectively optimize the code it generates
- ***Exception handlers:*** Deal with unusual situations and errors

These things are not unique to Java. Some are also characteristics of Smalltalk (developed in the early 1970s) and there are also close similarities to Microsoft's .NET CLR. Interestingly, at least one MRTE implementation supports both Java and Microsoft's Common Language Infrastructure (CLI), with only a relatively small amount of customization required to handle the different bytecode formats of CLI and Java.

MRTEs have emerged rather quietly. When Java was first released by Sun in 1995, most of us weren't really framing the benefits of Java in terms of managed code, rather we were struck by the promise of WORA and the benefit of a powerful JDK API, which made it pretty straightforward to start developing your own applications, even if you were new to the world of object orientation. Many of the intrinsic benefits of an MRTE were initially secondary to a new breed of programmer but proved key to Java's expansion to the server, where it became easier to write secure, robust, multithreaded, memory-intensive applications. It was soon also widely recognized that the abstraction required to handle an intermediate binary format need not come at the cost of performance, as MRTEs can use a dynamic code optimizer to react to the runtime characteristics of an application. Look at what Microsoft is doing with .NET, or the work Intel is doing to evolve its new range of 64-bit Itanium processors, and it's clear that the (inevitable) migration to MRTEs is widely recognized by both software and hardware vendors. In fact, the proliferation of MRTEs is seen by many as the biggest paradigm shift in software since the move from assembler to high-level programming languages.

For an MRTE to be as efficient, stable, and scalable as possible, it shouldn't be just a black box. We should be able to peer inside to make sure there are no obvious performance bottlenecks, monitor memory usage, etc. *To be viable in the enterprise, a managed runtime should also be manageable.*

Rather than just blindly executing bytecode, a JRE has the ability to add value by adapting its environment to the running application. A few examples of how this is possible are:

- ***Progressive levels of code optimization:*** Optimization requires processor time, so the JRE should focus its most aggressive optimization on the most frequently called methods.
- ***Dynamic expansion or contraction of the heap size, according to the needs of the application***
- ***Adapting the garbage-collection algorithm to the requirements and behavior of the application:*** A typical trade-off here is acceptable maximum pause time versus overall throughput.

Not surprisingly, there are limitations to the JRE's ability to adapt to the behavior of a running application. Also, hardware resources are always limited, of course, and programmers often introduce performance bottlenecks, which are hard to remove. To get the best performance and scalability from your Java environment, you'll need to:

1. Optimize your application code
2. Tune the JVM for the application and underlying OS and hardware

These two things are obviously not entirely independent, so it may be an iterative process to fully optimize your runtime system. Since the early days of "almost-black-box" JVMs, various tools and techniques have evolved to help optimize your application during development and monitor your application in production. These tools generally use bytecode instrumentation or the Java Virtual Machine Profiler Interface (JVMPI). Several commercial tools are aimed at monitoring J2EE applications, e.g., Introscope (Wily Technologies), PerformaSure (Quest Software), and Cyanea/One (Cyanea).

Keep in mind that all of these tools introduce a performance overhead into your system – the more information you want, the higher the overhead – and they don't tell you much about the behavior of the JVM. To tune the JVM you'll probably benefit from running your application under load, looking closely at the performance and behavior of the application, and doing some trial-and-error adjustment of the available JVM parameters (which vary across different JVM implementations).

The BEA JRockit Management Console is a tool designed primarily to monitor the runtime characteristics of the JVM (as opposed to the behavior of the application). The console can connect to one or more remote Java processes and be used to monitor many runtime statistics and parameters of the running JVM. Two benefits here for a production system are: (1) the monitoring overhead is insignificant (measured to be less than 1% for several different workloads), and (2) the console has a configurable event notification framework that can be used to notify an operator of certain warning events, giving them the opportunity to react to a situation before something more dramatic happens (e.g., a trigger can be set to identify a low available free memory condition, perhaps allowing an operator an opportunity to take corrective action and avoid the infamous OutOfMemory Exception).

The JRockit console shows important characteristics of the JVM, such as the heap and CPU usage. It also includes a method profiler and an exception counter. Figure 1 shows the memory panel of the management console. Most of this information is also available in the administration console of WebLogic Server (which uses a standard JMX interface to expose JRockit runtime statistics).

Future Directions

Many of the current profiling tools and techniques can be invaluable in tuning your application or JVM, debugging problems, and capacity planning. However, many of these approaches also have some significant drawbacks:

- Applying any of the profiling techniques causes a perturbation to the running system. You can't measure the system without having some impact on it (the inescapable Heisenberg Uncertainty Principle).
- Licensing costs.
- They work only with specific JVMs (e.g., JFluid or the JRockit class preprocessor).
- The tools lack an active feedback loop, i.e., these approaches are monitoring rather than true management solutions. For example, if an operator sees the JVM is running out of heap space in the JRockit Management Console, it's not currently possible to dynamically resize the heap.

Most of these tools also miss an essential part of the equation, as they focus on application performance and give you little indication of whether performance problems are caused by the JVM (or maybe the underlying OS). BEA has recently made public their JRockit Runtime Analyzer (JRA). This tool collects a wide variety of JVM runtime data for a specified sampling period. A graphical tool is then used to display this data, providing deep insight into the behavior of both the application and the JVM. One of the major benefits of this tool is that the profiling capability is built directly into the JVM, which allows for a much lower overhead (measured to be <1%) compared to the other common techniques of bytecode instrumentation, or JVMPI.

Another great benefit of JRA is that it collects a lot of data that's not possible with a standard JVMPI-based tool, e.g., JRA shows you which methods have been optimized and provides detailed information about all the GC cycles that have occurred during the profiling phase. The JRA is a very versatile tool that can add value in a variety of different situations:

- Developers can use the JRA to identify bottlenecks in their application, or provide data to help tune their heap and GC parameters.
- The BEA Customer Support team can analyze JRA profiles sent by customers who suspect a JVM problem.
- JRA data is used internally within BEA, in advanced development of the JRockit JVM.

More details on this tool can be found at http://edocs. bea.com/wljrockit/docs81/jra/jra.html.

In addition to vendor-specific solutions, various JSRs have emerged to evolve the JVM into a manageable environment and address some of the limitations of earlier implementations of the experimental JVMPI. J2SE 1.5 will include a new Java Virtual Machine Tool Interface (JVMTI), and an API to monitor the state and control the execution of programs running inside the JVM. JVMTI is designed to provide an interface for a wide variety of tools that need access to the VM state, e.g., profiling, debugging, monitoring, thread analysis, and code coverage tools. This work is being implemented through JSR 163 (and is closely related to JSR 174). JVMTI supercedes the Java Virtual Machine Debugger Interface (JVMDI) and prerequisites the new Java Platform Profiling Architecture (JPPA), which will supercede and improve upon the experimental JVMPI.

Group	Example Functions
Thread	*SuspendThread* *ResumeThread* ...
Heap	*ForceGarbageCollection* *IterateOverObjectsReachableFromObject* *IterateOverInstanceOfClass* ...
Stack Frame	*GetStackTrace* *PopFrame* ...
Object	*GetObjectSize* *GetObjectHashCode* *GetObjectMonitorUsage*
Breakpoint	*SetBreakpoint* *ClearBreakpoint*
Watched Field	*SetFieldAccessWatch* *ClearFieldAccessWatch* *SetFieldModificationWatch* *ClearFieldModificationWatch*

Table 1 Example JVMTI functions

JVMTI agents run in the same process as the JVM being examined, communicating through a native interface that's designed to allow maximum control while introducing minimal overhead on the running system. Agents should be as lightweight as possible, and are controlled by a separate process that implements the bulk of a tool's function without interfering with the target application's normal execution. JVMTI is a two-way interface whereby agents can both query and control applications through functions, and be notified of interesting events. JVMTI functions may be categorized into different groups. Some examples of available functions are given in Table 1.

Agents can respond to many different events that occur while running a Java application. Some example events that may be generated:

- ***Breakpoint:*** When the running application hits a predefined breakpoint
- ***Field Modification:*** When the application accesses a designated field
- ***Method Entry/Exit:*** When the application enters/exits a particular Java programming language or native method
- ***Class Load/Unload:*** Generated when a particular class is loaded/unloaded
- ***Garbage Collection Start/Finish:*** Generated when a full-cycle GC begins/ends

Stuart Lawrence is the program manager for BEA's Java Runtime Products Group. He was previously an engineering manager in the WebLogic Server team and prior to BEA he worked on JDK implementations at IBM and Sun Microsystems. He holds a PhD in physics from the University of Oxford.

stuartl@bea.com

Bob Griswold is the VP and general manager of the Java Runtime Products Group at BEA Systems, responsible for the overall management of the WebLogic JRockit JVM business. He has also held management positions at Sun Microsystems, as well as the Boston Consulting Group. He holds an MBA and a master's in East Asian studies from the University of Chicago.

griswold@bea.com

Profiling Your Java Code

Here's a quick survey of some of the tools and techniques available.

Bytecode Instrumentation (and Class Preprocessing)

Java bytecodes can be instrumented statically (i.e., by the generation of an instrumented .class file) or dynamically at runtime during the class-loading process. Many profiling tools use the latter approach, utilizing a custom class loader that modifies the byte stream read from the .class file during the class-loading process. Usually only certain classes will be instrumented, to minimize the performance overhead of the profiler.

The Byte Code Engineering Library (BCEL) provides freely available tools that can be used to instrument existing classes, or create them from scratch (http://jakarta.apache.org/bcel/). The BCEL API may be used to instrument classes both statically or on the fly. It offers a convenient abstraction that avoids the need to directly modify the bytecode yourself.

Another freely available tool is AspectWerkz (http://aspectwerkz.codehaus.org), an open source project sponsored by BEA's JRockit development team. AspectWerkz is a dynamic, lightweight, and high-performance AOP/AOSD framework for Java. It utilizes runtime bytecode modification to weave classes at runtime. It hooks in and weaves classes loaded by any class loader except the bootstrap class loader, with special support for runtime weaving using JRockit, allowing the addition, removal, and restructuring of advices as well as swapping the implementation of introductions at runtime.

Bytecode instrumentation is also used by many commercial tools such as Introscope from Wily Technology (www.wilytech.com). This tool provides a real-time performance monitoring solution for application servers. On-the-fly bytecode instrumentation is used to provide a lot of performance data that is not accessible through application server vendor–specific tools.

JFluid is an experimental dynamic bytecode profiler from Sun (http:// research.sun.com/projects/jfluid/). It can be used to profile an arbitrary subset of your Java program and be activated/deactivated while the program is running. Unlike the other tools above, it requires the use of a special version of the HotSpot VM that supports dynamic bytecode instrumentation.

The JRockit JVM provides an interface to make it easy for you to plug in your own class preprocessor. In other words, rather than requiring a special class loader, you register your own implementation of the com.bea.jvm.ClassPreProcessor interface, which has only one method:

```
public byte[] preProcess(java.lang.ClassLoader classLoader,
java.lang.String className, byte[] classBytes)
```

where the method parameters are the class loader instance used to load the class in question, the name of the class, and the byte array that represents the bytecodes of the class. A class preprocessor may be registered in the JVM instance using:

```
JVMFactory.getJVM().getClassLibrary().setClassPreProcessor
(preProcessor);
```

A simple example of how to use a preprocessor can be found at http://dev2dev.bea.com/products/wljrockit81/resources.jsp.

The Java Virtual Machine Profiler Interface

Many of today's Java profiling tools use the experimental Java Virtual Machine Profiler Interface. JVMPI defines a bidirectional interface between a JVM and an in-process profiler agent. This agent sends profiling data to the profiler front end, which displays information on CPU usage, memory allocation, object references, and monitor contention. JVMPI is used by both Borland's Optimizeit Profiler (includes a memory and CPU profiler) and JProbe from Quest Software.

Hprof (http://java.sun.com/j2se/1.4.2/docs/guide/jvmpi/jvmpi.html#hprof) is a JVMPI profiler agent shipped with many JDKs. It uses the JVMPI to gather information about a running JVM and writes out profiling information either to a file or to a socket.

HPjmeter from Hewlett-Packard is a useful free tool that allows you to read (and compare) the output files from hprof (www.hp.com/products1/unix/java/hpjmeter/index.html). Another useful tool from HP (not based on JVMPI) is HPjtune, which can be used to read the GC metrics in the log files produced by the Java VM (output of -Xverbosegc).

JVMTI includes an extension mechanism whereby a JVMTI implementation can provide functions and events beyond those defined in the specification. This mechanism can be used to provide JVM implementation–specific information, which may be invaluable for monitoring or debugging situations.

Also included in JSR 163 are APIs for JVM monitoring and management (java.lang.management) and in-process instrumentation (java.lang.instrument). The former API, described in more detail in JSR 174, provides a monitoring and management interface to the JVM and the operating system. It is designed to be suited to production environments and can be used to gather data on the threading, class loading, and memory subsystems, for example. The java.lang.instrument API, known as Java Programming Language Instrumentation Services (JPLIS), allows agents written in Java to instrument applications running on the JVM. The agent provides an implementation of the ClassFileTransformer interface, which is used to modify the bytecode of a class before it's actually defined in the JVM (i.e., providing a standard way to do class preprocessing). Details of the proposed "management" and "instrument" APIs may be found by downloading the latest version of the JPPA.

Summary

We've described some of the benefits of Java in the context of the JRE as a Managed Runtime Environment. We surveyed several tools and techniques available to profile Java code and monitor the runtime characteristics of a JRE. In the future, the manageability of JRE implementations will be enhanced through the standard JVMTI interface as well as through vendor-specific tools. Driven by the needs of server-side, business-critical applications, the Java managed runtime will evolve into a truly manageable runtime.

References

- *Intel Technology Journal*, Vol. 7, issue 1: www.intel.co.jp/technology/itj/ 2003/ volume07issue01/art01_orp/p03_mre.htm
- *JSR 163: Java Platform Profiling Architecture:* www.jcp.org/en/jsr/detail?id=163
- *"Using the Monitoring and Management APIs"*: http://edocs.bea.com/wljrockit/docs81/jmapi/index.html
- *JSR 174: Monitoring and Management Specification for the Java Virtual Machine:* www.jcp.org/en/jsr/detail?id=174

Due to delivery pressures on our last project,
we thought about moving to .NET.
I suggested we stick with Java, but use ThinkCAP.

Investing in 'Professional Open Source'

by Jeremy Geelan

EXCLUSIVE JDJ INTERVIEW WITH

DAVID SKOK

GENERAL PARTNER, MATRIX PARTNERS

This past February David Skok's new VC firm – Matrix Partners – orchestrated, with Accel, a $10 million investment in JBoss, Inc. This first round of funding in an open source company was a bold play, but then David Skok, famous in the Java arena as the founder of SilverStream Software – acquired by Novell in 2002 – is no stranger to bold moves.

The first question then, naturally enough, is to ask Skok whether this investment is somehow a contradiction, or whether there is indeed money to be made from open source?

"We think that there's definitely money to be made," Skok says without hesitation. "Red Hat's a clear example," he adds. "They're now profitable on around a $100m annual revenue run-rate. And we're still in the early days of the open source movement."

Curious about the business model he sees underlying OSS, I ask him how he usually explains it to fellow members of the investment community.

There are two business models, Skok explains, "dual license" and "support."

"In the dual license model," he continues, "followed by companies such as MySQL, software is available free with a restrictive GPL license, and available for a small fee with a less restrictive commercial license.

"You also see companies like SourceFire," Skok adds – they are the ones who developed Snort, the network intrusion detection system – "offering an upgrade product to Snort for a fee."

Then there is the second model, the support model. "JBoss follows the support model," Skok says. "What they call 'Professional Open Source'."

Customers Want a 'Throat to Choke'

There's one compelling reason why this model is so successful, Skok notes. "It's driven by a very important customer demand: the customer wants a 'throat to choke' when they move from development into production, and know that they can only get accountability and guaranteed response times if they pay for support."

Is "Professional Open Source" the wave of the future? I ask.

"Open source should be thought of as a different way of developing software," Skok replies. "With Professional Open Source, we should stop seeing open source developers as unemployed student types wearing Birkenstocks. Specifically, open source for middleware software makes a lot of sense because it's the most open/naked state of code and the peer review process leads to very robust, fast, and open stacks."

Given that open source is fast becoming a multimillion-dollar – most likely a multibillion-dollar – business, I ask Skok who the stakeholders are who can expect to benefit.

"I think that the greatest beneficiaries are customers and independent software vendors (ISVs) who will benefit from high-quality, royalty-free platforms," he replies.

Skok then adds a thought about profitability, for open source and profit are not mutually exclusive it seems. "One of the things that impressed us most about JBoss," he says, "is that it has been profitable since the beginning, and expects to continue operating that way into the future."

The $10 million capital was primarily raised to strengthen their balance sheet, Skok explains, reassuring customers that they are dealing with a company that is here for the long term. "They will also use the capital to expand operations into more regions and scale up support capabilities," he adds.

On a technical note, while JBoss has always been applauded for being robust and scalable, it's never had a GUI-based IDE, thus leaving the mass marketplace solely in the hands of closed source competitors. So I ask Skok what he thinks about the recent idea of plugging JBoss into the Eclipse development framework, making the product a legitimate option for the thousands of developers who prefer a GUI. Will that transform JBoss's fortunes from niche player to mass-market player, does he think? Does that partly explain the timing of this investment?

"In today's market," Skok answers, "it's no longer necessary to get GUI tools and an app server from one company. In addition to the Eclipse project, JBoss is already integrated in all leading IDEs: JBuilder X, Compuware OptimalJ 3.0, Eclipse, IntelliJ, and so on.

"I keep speaking to new startups that are coming out with hot new GUI-based development tools for things like UI development, and building XML services-oriented applications," he continues, "and they all tell me that they are planning to ship with JBoss as the default application server. They tell me that they initially picked JBoss because it's royalty free, but they then grew to love it because it's highly stable and has a very compact, modular architecture.

"The surveys that we discovered during our due diligence," Skok adds, "showed that JBoss was number one in terms of both developer and OEM/ISV usage, and had grown its enterprise market share to 27%. In the last year, they had grown their market share more than twice as fast as either BEA or IBM. They have managed to attract an incredible base of ISVs including companies like Apple Computer, Sterling Commerce, and Mercury Interactive to name just a few. Those were some of the key factors behind our decision to invest."

'Disruptive' Open Source Is the Future

Is the JBoss, Inc., business model, though – i.e., charging for support – the perfect business model, or does Skok expect to need to tweak it moving forward to ensure that the Matrix/Accel $10 million is leveraged to the max?

"We believe that their business model is the right one for today and the future," he maintains. "It's totally in tune with what customers want, which is low ownership costs for software, and a fully accountable vendor for support."

> "Is there money to be made from open source? 'Dual license' and 'Support'"

It's also a very nice business model financially, he adds, as support contracts are generally renewed, "so you end up with a very predictable revenue stream that only grows over time."

However, as with any business, Skok points out, "It's important to keep listening to the customers to see if there are tweaks that are needed going forward."

What about the wider picture, beyond JBoss – does Skok expect a wave of VC money to now pile into open source in the next five years in the same way as it did, say, to the wireless space in the past five years? Or does he think it will be an altogether more cautious process, more on a case-by-case basis?

"Open source is very hot right now," he says, "driven by the clear success of Linux in the enterprise. We have IBM to thank for that; I've heard that they've committed $2 billion to Linux marketing."

Wishing to further explore the whole Linux phenomenon, I ask him why he thinks it has taken off so astonishingly. He explains that the key to it, according to the customers he speaks to, is the financial attraction of escaping lock-in. "Customers tell me that they're attracted to Linux because of 'Intel Economics' – the ability to ride the price curve of Intel-based computing, as opposed to being locked into proprietary Unix platforms like Sun," Skok says.

"The initial open source successes are Linux and Apache," he continues. "However, customers are looking at the rest of the infrastructure stack and realizing that there are the same cost and lock-in avoidance benefits to be had."

Skok ranks JBoss as one of the four greatest open source opportunities right now: "The two leading companies, behind Red Hat and SUSE, that will benefit next are JBoss and MySQL. There will be others, but it's my belief that only a few will be big enough for venture capital style returns."

What about the whole open source versus proprietary debate: Does Skok think it's operationally – and legally – feasible to commingle the two types of software? Or does he fear like many that we're just ushering in an era of energy-sapping litigation, as IP issues are contested by teams of lawyers in the courtrooms of America and indeed the world?

There are a couple of main bumps in the road, Skok reckons: one is the GPL and the other is the whole question of intellectual property. "Users of open source need to watch out for the GPL license, which can require that they open source their own software under certain circumstances," he notes. JBoss, Skok adds, uses LGPL (Lesser GPL), which eliminates this problem.

The SCO lawsuit, says Skok, highlights the second issue – whether you're protected from IP problems. "JBoss solves this problem by indemnifying its customers against those liabilities. Novell and HP have chosen to do the same thing for their customers buying Linux."

What does JBoss, Inc.'s, newfound "Professional Open Source" status do to the whole landscape – do proprietary app server giants, like IBM's WebSphere, BEA's WebLogic, and Oracle's IAS 10*g* need to worry that JBoss will make them redundant, I wonder.

"JBoss (and open source in general) is highly disruptive to the business models of those vendors," Skok replies. "The recent recession put an extreme focus on costs, and forced IT to look for ways to do more with less. In the late '90s it was cool to tell your friends about the latest Sun E10k box that you had purchased for over a million dollars. Now it's cool to tell them how much money you've saved your company by deploying Linux on commodity Intel boxes."

Skok tells me how he's spoken to several customers who had initially chosen WebLogic or WebSphere as their app server of choice, and then switched to JBoss "because of the extraordinary cost savings," as he puts it.

"Their developers had all recommended JBoss as their preferred product," he adds. "It took a budget squeeze for them to take a chance and try it, but once they saw how good the product was, they weren't going back."

Technology Startups: Then and Now

Much has changed in the world of technology and business since Skok founded SilverStream as "The eBusiness Platform Company," so I was interested to hear what he felt was different in the landscape for startups today compared to how it was then.

> "Open source should be thought of as a different way of developing software. With Professional Open Source, we should stop seeing open source developers as unemployed student types wearing Birkenstocks"

"When we founded SilverStream, it was right when enterprises were starting to use the Web for business purposes," he recalls. "There was untold excitement and promise, and the future was wide open.

"Enterprises felt a lot of pressure to move quickly or be 'Amazoned.' That led them to ignore their traditional purchasing habits and buy from startups to gain a time-to-market advantage.

"After the bubble," he continues, "the pendulum swung too far the other way. No one wanted to buy from startups, and all the shelfware had eroded trust in sales promises of magical business gains. The pendulum is starting to swing back to normal, but it's still an extraordinarily tough environment. Products have to provide really clear demonstrable value and significant ROI in a short period of time. Lots of reference-able customers are crucial, and frequently startups need a channel partner to help solve the 'safe choice' problem."

In those days, Skok recounts, it was essential to build market share quickly, at any cost: all focus was on revenue growth, and profitability was considered very unfashionable. "These days," he says, "the focus is on capital efficiency, building businesses with the minimum amount of capital." That means a strong focus on expense control. Rushing to get to market is less important than really getting the product right and making the customer totally satisfied.

"We're back to the days where it will take four to five years to build a real company, and the only people starting them are those with a true passion for the technology and their customers," adds Skok. "I've always believed that the only way to make money is not to focus on making money, but to focus on building real value and making your customers happy. In the bubble, we had lots of the wrong kinds of people starting companies, trying to make a quick buck."

So much for what's different. But there must be similarities too, "eternal truths" of the startup game, if you will.

Skok agrees. "What hasn't changed is that you still need to focus on building an exceptional management team consisting of only A players," he says.

"A players attract other A players; B players attract C players. The difference in performance between an A player and a B player is enormous, whether it be development, sales, or any other function.

"Over many years," Skok says, "VC firms have collected a lot of wisdom about what it takes to succeed. The key ingredients that we look for are:

1. An outstanding management team
2. A market that is feeling real pain, that is large and growing
3. Long-term sustainable differentiation/ barriers to entry

I ask Skok what was it like at first for him personally, transitioning from serial entrepreneur to VC? "I guess it was like moving from being a parent to a grandparent," he replies.

"As a parent you have constant day-to-day involvement with the child including all the highs and lows (changing nappies, etc.). As a grandparent, you're less involved, get to share in some of the fun, but also have less of the day-to day-stress. The biggest change is switching from making all the decisions to becoming an advisor and mentor to the CEO."

Technology Insights

He doesn't own businesses outside of technology. "I think my expertise is limited to the technology field, so other than professionally managed stock market holdings, I have stayed away from other investments. My job also requires total focus and doesn't leave much time for anything else."

So he doesn't object when I ask him to name one thing that Java has gotten completely right – and, conversely, one thing in Java's history he wished had turned out differently.

"Java got two things right," Skok says. "First, platform independence – which means the same app will run on any platform unchanged. And second, standardization – which means vendor choice and competition." I think that we'll continue to see both .NET and Java as competing platforms in the foreseeable future. Web services will greatly increase interoperability."

What's his response to those who argue that Sun has somehow lost its moral right to exercise exclusive control over Java and that Java ought now to be open sourced? "I'm in favor of open sourcing the source code for the Java reference implementations, but allowing Sun to retain the rights to the Java brand. Sun could still maintain control over the verification and certification of products based on them passing Sun's compliance tests. That model will dramatically improve the quality of Java, give developers the comfort that they are not playing in a totally Sun-controlled world, and have the ability to contribute to the reference implementation, while still leaving Sun a way to make money out of Java.

"In my opinion that would increase the acceptance of Java in the market, and reduce the chances of success for competitive languages like C#."

Beyond Java: Technologies to Watch

Wearing his new generalist hat as a general partner at Matrix Partners, Skok has a bird's eye view of hot investment areas beyond the Java world. Several areas are really interesting right now, he says: utility computing and datacenter virtualization, wireless and mobility, offshoring, RFID, and security, to name just a few.

> "I'm less hopeful for Java on the client so long as Microsoft continues to fight it. But I haven't written it off"

What Java has gotten wrong, in his opinion, is that "it has become too complex."

"That complexity is necessary in certain respects (as distributed applications are complex in nature). Expert developers appreciate the need for this. However, not enough attention has been paid to simplifying development, and making Java more accessible to a less expert audience who I believe make up at least 60% of the corporate developer audience."

The most promising answer to the overcomplication of J2EE, Skok adds, is something JBoss introduced in their 4.0 release: aspect oriented programming.

"AOP has the potential to greatly reduce the complexity, allowing developers to add capabilities like persistence, remoting, security, transactional integrity, and so on, to plain old Java objects (POJOs) without the complexity of today's EJBs. This is close to the .NET approach. JBoss is working hard on the EJB 3.0 specification to help bring in these kinds of advances. They're hopeful that that group will find a way to make enterprise Java easier to use. This is important to keep the overall market share of Java versus .NET."

In addition, Skok believes that the J2EE world needs high productivity GUI tools for less expert developers in the area of UI development. Things to help them graphically design page flows, and build data-bound forms and reports.

"Today .NET is the only serious challenger to Java in the enterprise," he says. "I'm less hopeful for Java on the client so long as Microsoft continues to fight it. But I haven't written it off.

"I'm personally very excited about the opportunities around the datacenter," Skok explains. "It's clear that today's datacenters are way too complex to manage, and can't respond fast enough to changing business needs."

The datacenter of the future, he believes, will be very different: there will be a big pool of storage, a large pool of processors, and a big network pipe coming into a lights-out facility.

"You'll only need to go in there to add new processors/disks or to remove broken units. New virtual computers will be created out of these pools via a simple GUI. These virtual computers will share resources from the pools, but will be isolated from one another. They will be dynamically allocated more or fewer resources as demand grows and shrinks. My investment play in this area is a company called Katana."

"RFID is also very hot right now," Skok says, "driven by a mandate from Wal-Mart that its top 100 suppliers have to start shipping them pallets that are identified by RFID tags.

"This was followed by a similar mandate from the Department of Defense. That has kicked off a mad scramble by their vendors to become RFID enabled. We've invested in a company called OAT Systems, and they are overwhelmed with customer demand."

Infosecurity is another interesting area, he says. "One of the driving forces is regulations such as HIPAA, GLBA, etc. We're seeing a fair number of business plans focused on auditing and protecting access to information assets in databases, files, and so on."

Skok has also spent a lot of time investigating grid computing. "I'm less enthusiastic in the short term, though," he says. "Grid computing today works best for technical computing applications where the workload can be easily split into modules that can be run remotely in parallel. It also requires that applications be rewritten to support a Grid API (OGSA being the key standard).

"When I look at enterprises, I see mostly applications that cannot be run remotely as they all require access to a single database, and I see a huge reluctance to rewrite applications. What's more interesting to enterprises is the utility computing model where they can run these applications unchanged on a shared infrastructure that responds dynamically to changing workloads. That will be a much bigger market in the short to medium term."

Over time, grid computing could become more interesting, he conceded – "as we move to highly modularized applications built out of Web service components."

Skok ends by stating how VCs feel about the current environment, after going through the tech recession of the last few years.

"There's a definitely increased optimism right now. We're starting to see companies meeting or beating their plans once again. As an example, one of my portfolio companies, Netezza, had a very aggressive forecast for its fourth quarter, and was able to bring in bookings that were double that target. We haven't seen that in a long time.

> "The datacenter of the future will be very different: there will be a big pool of storage, a large pool of processors, and a big network pipe coming into a lights-out facility"

"The public markets are getting better, and will be helped by the IPOs of Google and Salesforce.com. And there is a lot of M&A activity.

"Off-setting that positive news," Skok adds, "we still have an environment where there is too much venture capital chasing too few deals; an overhang from the bubble. This means that there is overfunding in hot areas. As an example, there were no less than 64 companies building WiFi chips when we last counted. I predict that in the long run only the top-tier venture firms will make money and survive. They have access to the best deals, and have the most experience in helping those companies build for success and avoid pitfalls."

Finally, is there such a thing as an "ultimate software solution" – or does Skok view that as a mere mirage?

"I guess it would have to be a system where you could think at it, and have it interpret your thoughts to build fully functional, scalable, reliable, secure applications automatically," he says. "It would also want to behave more like biological systems: i.e., be autonomic (self-healing), able to detect changing needs in the environment, and suggest ways to evolve itself.

"Is that a mirage? For the time being, I guess so…"

The Perils of Copy-Paste Coding

Three approaches that help make it work

Paul Mukherjee

Copy-paste coding is a kind of misguided code reuse. You have a problem to solve and you see a similar problem and its solution in your existing body of code. So you copy and paste the solution, and make the necessary modifications so that the solution matches your current problem.

Here's an example to make this more concrete: you've written a system that allows departments in your organization to analyze their productivity; each department has its own ideas about what it wants, so each has its own domain logic. The sales department wants to be able to export data from the system into a planning tool. After a few months in production, the personnel department spies this particular feature and says they would like it as well. No problem, says your boss; we already have that functionality in the system so we can make it available to you in the next maintenance release. Your boss then arrives at your desk one morning asking you to make this modification, assuming it won't take long because you already have the code.

The problem is that functionality, which is intended to be specific to one business domain, is not usually written generically. The ideal solution would be to rewrite the code to export to a planning tool as a generic component that can be configured for multiple business domains. However, doing it properly means you can't make the deadline that your boss has promised to the customer. Your only option is to take a copy of the existing code, say as a new class, and then modify that copy for the personnel department's specific requirements. This results in two different classes that share a similar functional structure, but whose details vary according to the business domain.

Paul Mukherjee is a Sun Certified programmer and developer. He works as a consultant software architect for Systematic Software Engineering in Denmark.

pmu@systematic.dk

What's the Problem with Copy-Paste Coding?

It's a bit ugly, but is there any real problem with copy-paste coding? Yes! The list is long, but I'll concentrate on the two major issues. Each time you copy something, you are adding something to your maintenance burden. In the above example, if the planning tool was upgraded, in the worst case scenario (which in my experience is the normal case!) each copy of the exporting code would need to be modified. If the organization has standards concerning test coverage, these code repetitions would require individual testing. These are all very practical concerns for a developer.

There is also an issue of principle: tailoring the export code for each business domain leads to tight coupling between the export code and the business domain. Changes to the business domain could have unexpected repercussions for the exporting feature.

What Can You Do About It?

Are we stuck with multiple copies of very similar code? Is the only alternative a full-blown refactoring of the code into a generic component? No! There is a middle road that ensures that only one version of the repeated code exists. With a decent refactoring tool such as Eclipse's JDT or IntelliJ's IDEA, it's easy to refactor the existing code into a maintainable, loosely coupled version in a relatively short time. The approaches I'll describe have these nice properties without having the flexibility that a more generic component might have. In my experience, there are three overall solutions; in practice a combination of these solutions is often needed.

All three solutions rely on analyzing corresponding methods. I often find that the easiest way to do this is to print out the relevant methods and then look at them side by side. This way it's easier to see which lines are common to both methods and which lines are domain specific; I usually draw boxes around the domain-specific lines.

Solution 1: Helper Classes

If the two methods contain similar or identical nondomain-specific code, they can be moved to a helper class. Continuing the earlier example, two different departments in an organization export data to the same planning tool, based on their own information. The personnel department's export is implemented using the class PersonnelInfo Exporter shown in Listing 1. (Listings 1–6 can be downloaded from www.sys-con.com/java/sourcec.cfm.) The sales department's export is implemented using SalesInfoExporter, shown in Listing 2. The details of the planning tool are not really important to this article.

Looking at these two classes, the similarity between the two export methods is quite striking. The structure for both is identical: initialize the tool, populate the tool with data, and terminate the tool. For PersonnelInfo Exporter the three tasks are, respectively, on lines 18–22, 23–64, and 65–67; SalesInfoExporter has these tasks on lines 16–20, 21–51, and 52–53.

Refactoring

To quote Martin Fowler, "Refactoring is the process of changing a software system in such a way that it does not alter the external behavior of the code, yet improves its internal structure." In practice, this means taking something that works and improving its design so that it's easier to maintain, extend, debug, and so on. In this article I refer to a number of standard refactorings, details of which can be found at the refactoring Web site in the references list.

This structure suggests that the first and third tasks have little to do with the actual domain-specific data being exported, so we should be able to extract methods for initializing and terminating the tool. A refactoring tool should make this straightforward. In this case, the application of the refactorings Extract Local Variable, Extract Method, and Inline Temp leads to two methods:

```
public PlanningTool
  initializeTool(String title,
                 int numColumns) {
  PlanningTool planningTool =
    PlanningTool.openConnection();
  planningTool.createChart(title,
                           numColumns);
  return planningTool;
}

public void terminateTool(
  PlanningTool planningTool,
  String footer) {
  planningTool.setFooter(footer);
  planningTool.closeConnection();
}
```

Since neither of these methods is specific to a domain or uses instance variables, they can be moved as static methods to a helper class, in this case ExportHelper, as shown in Listing 3. The new SalesInfoExporter is also shown in this listing; similar changes would be made to PersonnelInfoExporter. These methods are static as it makes their state-independence explicit.

Solution 2: Inheritance

If two methods contain several similar portions of domain-specific code, I use the Template Method pattern by applying the Form Template Method refactoring.

The first stage in this refactoring is to extract the domain-specific parts of the method into separate methods. For instance, if we look at the SalesInfoExporter in Listing 3 we can see that its algorithm is as follows:

1. Initialize the tool with the title and number of columns (lines 24–26)
2. Fetch data for the period (lines 27–28)
3. For each year in the period:
   - Create an empty list (line 34)
   - For each column, add the data item for the column to the list (lines 35–53)
   - Add the data series consisting of this list and year to the tool (lines 54–56)
4. Terminate the tool with the footer text (lines 58–59)

Template Method Pattern

The Template Method pattern is one of the original Gang of Four design patterns (*Design Patterns: Elements of Reusable Object-Oriented Software* by Erich Gamma, et al). This pattern captures common functionality in an abstract superclass, and domain-specific functionality is located in concrete methods in the subclass known as hooks. These hooks are defined as abstract methods in the superclass and can therefore be referred to from there. See the references at the end of the article for more information about this very useful pattern.

Those parts of the algorithm that are domain specific are underlined and will be refactored as methods using the Extract Method refactoring. For example, I'll create a method getTitle():

```
protected String getTitle() {
  return "Sales Statistics for Period ";
}
```

The corresponding part of the export method is:

```
PlanningTool planningTool =
  ExporterHelper.initializeTool(
    getTitle() + startYear + "-" +
endYear,
    getNumberOfColumns());
```

When doing this type of refactoring, I try to use method names that indicate the role performed by these lines in the original methods. Using this approach, it's easy to extract the methods getTitle(), getNumberOfColumns(), and getFooterText().

However, this still leaves the body of the loop unresolved; what can we do about getting the data item for the current column? One approach would be to create a method called getDataItem (Object obj, int column), which takes the object currently iterated over and generates the corresponding data item for the column. This would work, but in my experience working with Object instances and then upcasting to the class we are interested in indicates that the design lacks something. In this case it's most natural to create an interface representing an object that can be exported:

```
public interface IExportableData {
  public int getColumnCount(int column);
  public String getTitle(int column);
  public Color getColor(int column);
}
```

I can then create a PlanningTool.DataItem object from an IExportableData object. There is now a design issue: Does the export of data fall under the responsibility of SalesInfo and PersonnelInfo, respectively? This depends on the specific applications; if it does fall under their responsibility, it's appropriate that the corresponding classes implement IExportableData directly. Otherwise it's more appropriate to create implementation classes of IExportableData (e.g., ExportableSalesInfo and Exportable PersonnelInfo), which, respectively, delegate to SalesInfo and Personnel Info. In this example I've chosen the former solution; for example, the new SalesInfo class can be seen in Listing 4.

With this interface in place, the last piece of domain-specific functionality is the retrieval of the list of data for the given domain. We create a hook for this that the implementations use to call the corresponding controller. For example, for SalesInfo:

```
protected List getDataForPeriod(
  int startYear, int endYear) {
  return new SalesController().
    getSalesInfo(startYear, endYear);
}
```

After this refactoring, exportSales Info() contains no domain-specific code. The approach now is as follows:

1. Create an abstract superclass (Abstract Exporter); domain-specific subclasses (SalesInfoExporter and PersonnelInfo-Exporter) should extend it.
2. In the abstract superclass, create abstract methods for each hook (get Title(), getColumnCount(), getDataFor-Period(), and getFooterText()).
3. Move the method (exportSalesInfo()) from the subclass to the superclass, possibly renaming to remove domain-specific connotations.

That's it!

AbstractExporter is shown in Listing 5 with the refactored SalesInfoExporter class.

Note that sometimes the renaming part in stage 3 might not be possible (for example, if a particular interface is to be implemented or preserved). In this case the method should just call the superclass method, as shown below:

```
public void exportSalesInfo(
  int startYear, int endYear){
  exportInfo(startYear, endYear);
}
```

This refactored design is a major improvement over the original one: there is now very little repetition, the separation of responsibilities is much clearer, and the code performing the export and the data to be exported are now loosely coupled.

One practical issue that often arises is that it isn't always possible to choose the superclass of a class; for example, if the class has to extend a specific superclass to fit into a particular framework. In this case we choose solution 3.

Solution 3: Delegation

This solution is very much a variation of solution 2 but instead of using inheritance, delegation is used. There are four steps.

First, an interface defining the hooks described in the previous section (lines 36–41 in Listing 5) should be created:

```
public interface IExporter {
  public String getFooterText();
  public List
    getDataForPeriod(int startYear,
                       int endYear);
  public String getTitle();
  public int getNumberOfColumns();
}
```

Next, the classes SalesInfoExporter and PersonnelInfoExporter should implement this interface; the implementations from solution 2 (e.g., lines 46–59 in Listing 4) should be made public. The third step is that the abstract superclass from solution 2 should be concrete and have an IExporter instance variable. The hooks that were previously in this class should be removed, and calls to them replaced by calls to the corresponding methods on the IExporter instance variable. This is shown in Listing 6.

Finally, the export methods in SalesInfoExporter and PersonnelInfoExporter should create an instance of AbstractExporter and call its exportInfo() method. For example:

```
public void exportPersonnelInfo(
  int startYear, int endYear){
  new AbstractExporter(this).
    exportInfo(startYear, endYear);
}
```

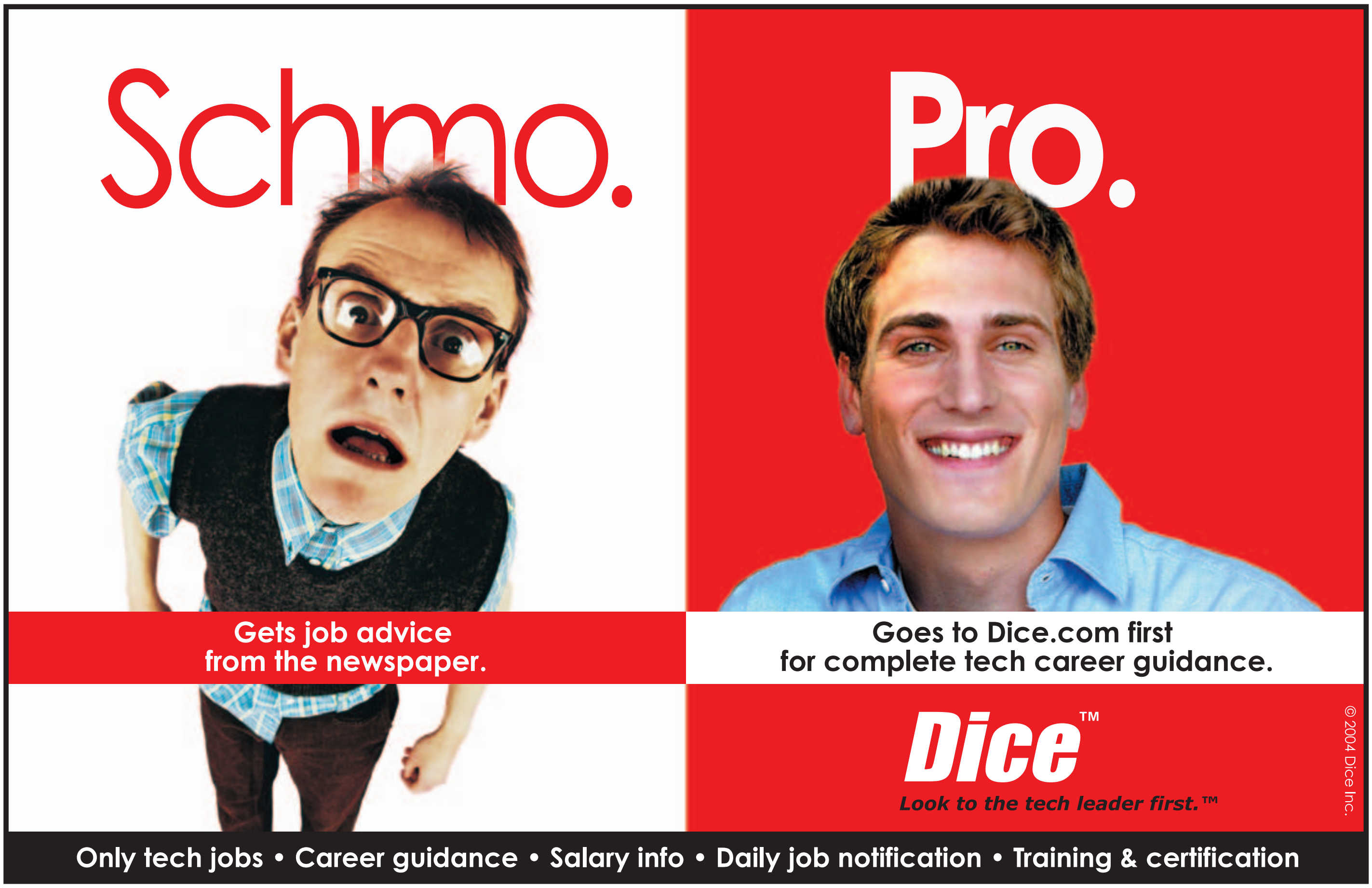
SalesInfoExporter and PersonnelInfoExporter could be further decoupled from AbstractExporter using an Inversion of Control pattern (see the list of references for more details). The idea behind solution 3 is exactly the same as solution 2. However, their structures differ according to whether inheritance or delegation is more suitable.

Closing Remarks

The techniques described in this article are practical ones that I have found useful on a number of projects. Copy-paste programming is easy and helps to provide something that works, fast. However, as soon as the first copy-paste phase is complete, it's important that the techniques described in this article are applied before the problem with repetition gets out of hand; the sooner the problem is addressed, the easier, quicker, and cheaper it is to resolve.

References

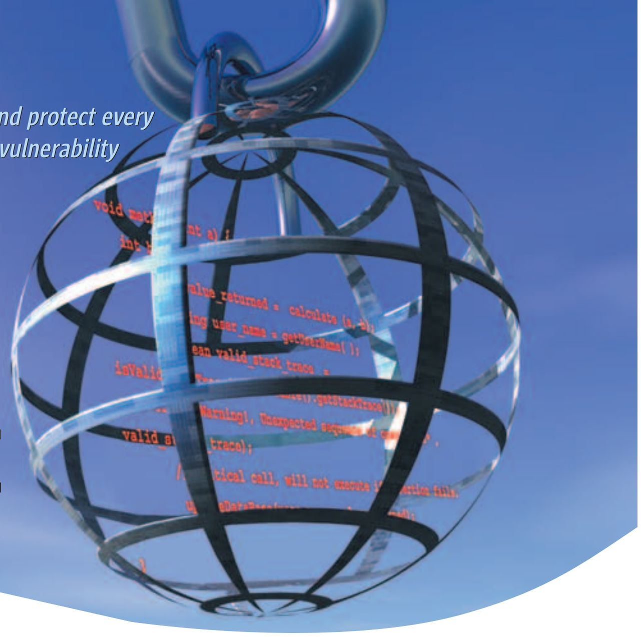
- *Refactoring:* www.refactoring.com
- *Template Method Design Pattern:* www.javacamp.org/designPattern/template.html
- *The IDEA Tool:* www.intellij.com/docs/IDEA_30_Overview.pdf
- *Eclipse's JDT Tool:* www.eclipse.org
- *Refactoring in Eclipse:* www-106.ibm.com/developerworks/opensource/library/os-ecref/
- Hammant, P. "Inversion of Control Rocks." ***Java Developer's Journal***, Vol. 8, issue 12.


Schmo.
Pro.
Gets job advice
from the newspaper.
Goes to Dice.com first
for complete tech career guidance.
Dice™
Look to the tech leader first.™
© 2004 Dice Inc.
Only tech jobs • Career guidance • Salary info • Daily job notification • Training & certification

Feature

Anticipate and protect every possible security vulnerability

STRATEGIES FOR SECURING JAVA CODE



by Adam Kolawa, Gina Assaf, and Roberto Scaramuzzi

Java security is an overwhelming issue. For a truly secure application, you need to prevent hackers from entering the system, and you need to ensure that code safeguards security if a hacker does break in. Moreover, there is no room for error. If you anticipate and prevent hundreds of security vulnerabilities but overlook just one, a hacker can still wreak havoc on your system.

This article introduces some fundamental strategies for writing Java code that remains secure if a hacker manages to enter the system. Essentially, writing secure code requires a shift in thinking. Instead of worrying about whether code works correctly, you need to anticipate all of the ways that it can be exploited, then ensure that security is maintained in every possible worst case scenario. This, of course, is a monumental task, and there is no silver bullet for security. Several strategies for developing Java code that resists many common attacks are:

- Follow coding guidelines for preventing security attacks
- Code defensively to prevent hackers from propagating through the code
- Expose and correct as many bugs as possible

Dr. Adam Kolawa cofounded Parasoft with a group of fellow CalTech graduate students in 1987. Parasoft provides Automated Error Prevention solutions that combine advanced tools, services, and expertise that help companies to automatically prevent errors and improve software quality and reliability. Adam holds a PhD in theoretical physics from the California Institute of Technology.

ak@parasoft.com

Follow Coding Guidelines to Prevent Security Attacks

Ensuring that your code complies with the following coding rules can make your code less vulnerable to security attacks.

Rule 1: *Avoid using inner classes*

Java bytecode has no concept of inner classes, so the compiler translates inner classes into ordinary classes that are accessible to any code in the same package. An inner class gets access to the fields of the enclosing outer class – even if these fields are declared private – and the inner class is translated into a separate class. To let this separate class access the fields of the outer class, the compiler silently changes these fields' scope from private to package. As a result, when you use inner classes, not only is the inner class exposed, but the compiler is silently overruling your decision to make some fields private.

Note: If you really need to use inner classes, be sure to make all inner classes private.

Here is sample code that violates this rule:

```
package examples.rules.security;

public class AUIC {

    private class AUIC2 {  // VIOLATION
    }
}
```

To correct this code, make the class AUIC2 a top-level class as follows:

```
public class AUICF {

}

class AUIC2 {  // FIXED

}
```

Rule 2: *Avoid comparing class objects by name*

This rule prohibits comparing class objects with the getName() method. More than one class in a running JVM may have the same name. As a result, a hacker can create a class with malicious code and give it the same name as your class. When you compare classes by name, the comparison would not recognize this difference. When you compare classes by object equality, the difference would be detected.

Here is sample code that violates this rule:

```
package examples.rules.security;

public class CMP {
    public boolean sameClass (Object o) {
        Class thisClass = this.getClass();
        Class otherClass = o.getClass();
        return (thisClass.getName() == otherClass.getName());
//VIOLATION
    }
}
```

To correct this code, modify it as follows to directly compare thisClass and otherClass for equality:

```
package examples.rules.security;
```

```
public class CMP {
    public boolean sameClass (Object o) {
        Class thisClass = this.getClass();
        Class otherClass = o.getClass();
        return (thisClass == otherClass);  // FIXED
    }
.}
```

Rule 3: *Make your classes noncloneable*

This rule requires that you make classes noncloneable by defining a final method clone that will throw a java.lang.CloneNotSupportedException(). For example:

```
public final Object clone() throws
java.lang.CloneNotSupportedException {
  throw new java.lang.CloneNotSupportedException();
}
```

Java's object cloning mechanism can allow an attacker to manufacture new instances of classes that you define, without executing any of the class's constructors. Even if your class is not cloneable, the attacker can define a subclass of your class, make the subclass implement java.lang.Cloneable, then create new instances of your class by copying the memory images of existing objects. By defining the above clone method, you'll prevent such attacks.

Note: If you really need to make your classes cloneable, be sure to make your clone() method final.

Rule 4: *Make your classes nonserializable*

This rule requires that if you want to prevent access to the internal state of your objects, make your class nonserializable. If you don't, your objects can be serialized into readable byte arrays. As a result, hackers can view the objects' full internal state, including private fields and the internal states of referenced objects.

To make a class nonserializable, define the final method writeObject() that throws an IOException. For example:

```
private final void writeObject (ObjectInputStream in)||    throws
java.io.IOException {
      throw new java.io.IOException ("Class cannot be serialized");
}
```

This method is declared final so that a subclass defined by a hacker adversary cannot override it.

> "Java code that throws unexpected, uncaught runtime exceptions is especially ripe for security breaches"

Note: If you need to make a class serializable, follow these tips to safeguard security:

- Use the Transient keyword for fields that contain direct handles to system resources and other sensitive information.
- Do not pass an internal array to any DataInput/DataOutput method that takes an array when defining your own serializing method for a class.

The source for these tips is "Secure Programming for Linux and Unix HOWTO" by David Wheeler (www.dwheeler.com/secure-programs/Secure-Programs-HOWTO/java.html).

Rule 5: *Do not depend on package scope*

This rule prohibits classes with public or package-private access. An attacker can simply add another class to your package and then access package-private fields that were supposed to be hidden.

To correct violations of this rule, modify code so that it does not rely on package-level access. Give your classes, methods, and fields the most restricted access possible. If this is not an option, you might want to use package sealing, which can prevent hackers from adding classes to a package that is in a "sealed" JAR file. Package sealing is discussed at http://java.sun.com/j2se/1.3/docs/guide/extensions/spec.html#sealing.

Rule 6: *Avoid returning references to mutable objects and, if necessary, clone the mutable object before returning a reference to it*

Mutable objects are objects whose states can be changed. When you return a reference to a mutable object, the caller can change the state of that object; if that object is used in the class later on, it will be using a different state of the object, which would affect the internals of the class.

Note that arrays are mutable (even if array contents are not mutable), so don't return a reference to an internal array with sensitive data.

Here is sample code that violates this rule:

```
// fDate is a Date Field.

// The caller of this method can change the Date object and
// affect the internals of this class.
public Date getDate() {
    return fDate;
}
```

Gina Assaf is a systems engineer at Parasoft. She has been developing, designing, testing, and implementing applications in Java for nearly six years. She has researched and developed many of the coding standards, many of which provide security for Java applications. Assaf has a BS in computer science.

Roberto Scaramuzzi is a software engineer at Parasoft. He grew up professionally as a Perl developer, but is now also adept at Java and PHP. Roberto holds a PhD in mathematics.

To correct this code, modify it to return a defensive copy of the field as follows:

```
public Date getDate() {
    return new Date(fDate.getTime());
}
```

Now, the caller of this method can change the returned Date object without affecting the internals of this class.

Rule 7: *Make methods, fields, and classes private; if there's a reason one should not be private, document that reason*

If a class, method, or variable is not private, hackers could use it as a potential entry point. If there is a good reason why a method, field, or class should not be private, it does not need to be private, but that reason should be clearly documented.

Rule 8: *Make all classes and methods final; if there is a reason one should not be final, document that reason*

If a class or method is not final, hackers could try to extend it in an unsafe way. If there is a good reason why a method or class should not be final, it does not need to be final, but that reason should be clearly documented.

Coding Defensively to Prevent Hackers from Moving Through the Code

Another way to boost code security is to code defensively by implementing defensive software firewalls and validating user/external inputs.

Design by Contract

Design by Contract (DbC) can be used to validate user inputs as well as to implement firewall-like restrictions that prevent hackers from moving through the application. DbC was designed to express and enforce contracts between a piece of code and its caller; these contracts specify what the callee expects and what the caller can expect.

Typically, DbC is implemented by expressing the code's implicit contracts in terms of assertions. Three types of conditions commonly used to create contracts are:

- ***Preconditions:*** To express conditions that must hold true before a method can execute
- ***Postconditions:*** To express conditions that must hold true after a method completes
- ***Invariants:*** To express conditions that must hold true any time a client can invoke an object's method

Java tools that support DbC generally have you incorporate specification information into comment tags and then instrument the code with a special compiler to create assert-like expressions out of the contract keywords. When the instrumented code is executed, contract violations are typically sent to a monitor or logged to a file. The degree of program interference varies. You can often choose between nonintrusive monitoring (problems are reported, but program execution is not affected), having the program throw an exception when a contract is violated, or hand-coding custom actions.

To learn more about DbC, see the following resources:

- Plessel, T. "Design By Contract: A Missing Link In The Quest For Quality Software": www.cs.unc.edu/~smithja/MIMS/DataModel/research/DBC.html
- *Interactive Software Engineering. Building Bug-Free O-O Software: An Introduction to Design by Contract:* www.eiffel.com/doc/manuals/technology/contract/page.html

Defensive software firewalls have many uses, one of which is to ensure that any hacker who manages to access your code cannot move through critical code sections. A defensive software firewall is a point in the logical flow of a program where the validity of logical constraints is checked. If a constraint specified in a firewall is satisfied, execution proceeds. If the constraint is violated, the firewall triggers, generates an appropriate error message, and possibly takes damage-control actions (such as stopping execution) or diagnostic actions (such as turning on debugging information). Firewalls should not alter the execution flow of a program unless they are triggered, and they should have either no effect or a negligible effect on the performance of the production version.

Note that these firewalls are not related to the firewalls placed around Internet connections. The concept discussed here relates solely to code construction. If designed carefully, these firewalls serve the same purpose as physical firewalls: they can contain a dangerous event and prevent it from causing a complete disaster.

In Java, firewalls can be implemented by adding assertions that say that if the program reaches a critical point without executing the expected operations, the program should throw exceptions, shut down, or perform another defensive action. Hackers often access code from a path that you didn't know existed or did not expect to be taken. By using firewalls to check whether critical program points were reached in unexpected ways, you can identify many attempted security breaches and stop hackers from entering and manipulating your most critical code.

To implement these firewalls with assertions, first determine the critical points in your code, then add assertions so that when each critical point is reached, the call stack is checked to verify that the program reached that critical point by executing the expected sequence of operations. When a critical point is reached through an unexpected sequence of operations, the program should throw an exception and terminate. Once this firewall is implemented, you will have a barrier that prevents hackers from using "backdoor paths" to gain entry into your most critical code.

For example, the code in Listing 1 could be added to verify the call stack before modifying a field in the database. In this case, the assertTrue method is assumed to accept an error message and a boolean. If the boolean is false, the message with an exception will be reported and the program will terminate.

Another defensive coding strategy is to validate all user/external input. When security is an issue, you should always worry about external inputs. If users manage to submit the "right" inputs, they could gain access to program details you hoped to keep private, prompt the application to crash or enter an unstable state, or access and modify the database. These inputs could be created by hackers trying to design inputs that cause a security breach (for example, by applying a technique known as SQL injection, where hackers submit inputs designed to create a strategic SQL string, such as a string that disables password checking, a string that adds a new account, etc.). Moreover, these inputs could also be submitted by well-meaning users who entered information incorrectly (as a result of a typo or a copy/paste error) or who simply misunderstood what type or format of information they were supposed to add.

By validating all external inputs, you can identify improper or malicious inputs as they are submitted and then prevent the program from using those inputs. The performance impact and user inconvenience will be minimal, especially if you consider the potential impact and inconvenience of having the application unavailable or functioning incorrectly because a hacker designed and successfully submitted the “perfect” input.

User input validation could be performed with assertions, Design by Contract, or if statements. For example, say you have the following method that accepts a filename as input from a user, but does not verify whether the filename represents a valid file.

```
void method (String filename) {
    System.exec("more " + filename);
}
```

If the user passes filename = "joe; /bin/rm -rf /*", the execution of that method could delete files. If the application runs as a regular user, the spawned process (rm -rf) inherits its privileges, which means that the user can remove only the files for which he or she has modify permissions. If the application runs as root, the spawned process will inherit root privileges and all files on the hard drive can be removed.

To prevent hackers from performing such disastrous deletions, this method should verify whether the entered filename represents a valid file. This verification could be implemented as a simple check using an if statement as follows:

```
void method (String filename){
    if (new File(filename).exists()){
        System.exec("more " + filename);
    }
}
```

Expose and Correct as Many Bugs as Possible

Every bug that you are not aware of could present hackers with an opportunity to exploit your system. Each bug represents unexpected program behavior; if the program is not behaving as you expect, how can you rest assured that it is not providing hackers with a way to manipulate your code?

Java code that throws unexpected, uncaught runtime exceptions is especially ripe for security breaches. Imagine that when one of your methods receives an input that you did not expect, it throws an uncaught runtime exception that’s thrown up several layers in your call stack before it’s handled, and the handling code exposes some critical Java code. If a hacker can produce this exception (for example, by flooding that method with a wide variety and range of inputs), he or she can then learn about the critical code.

The best way to expose these types of problems is to perform what is known as “white-box testing” or “construction testing.” This testing involves designing inputs that thoroughly exercise a class’s public methods, then examines how the code handles the test inputs.

For example, if you wanted to check whether uncaught runtime exceptions could cause a class to expose critical code in methods you think are protected, you would flood the class's exposed methods with a wide variety of inputs to try to flush out all possible exceptions, examine how the class responds to each exception, and modify the code so that it does not throw dangerous exceptions.

If you find errors during the testing phase, you have the opportunity to fix the problem before an actual security breach occurs. In fact, this type of testing is best performed at the unit level because testing each unit (each class) in isolation is the best way to expose the maximum number of errors. Because unit testing brings you much closer to the errors, it helps you detect errors that application-level testing does not always find. Take a closer look at Figures 1 and 2 to see how unit testing does this.

Figure 1 shows a model of testing an application containing many units. The application is represented by the large oval, and the units it contains are represented by the smaller ovals; external arrows indicate inputs; starred regions show potential errors.

To find errors in this model, you modify inputs so that interactions between units force some units to hit the potential errors. This is incredibly difficult. Imagine standing at a pool table with a set of billiard balls in a triangle at the middle of the table, and having to use a cue ball to move the triangle's center ball into a particular pocket – with one stroke. This is how difficult it can be to design an input that finds an error within an application. As a result, if you rely only on application testing, you might never reach many of your application's units, let alone uncover the errors they contain.

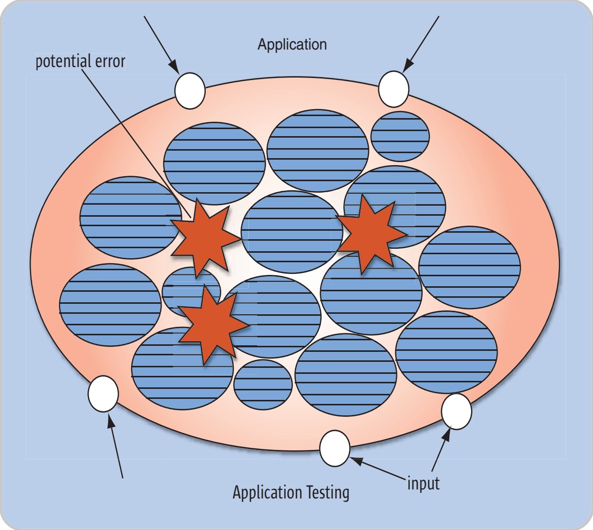


Figure 1 Application testing

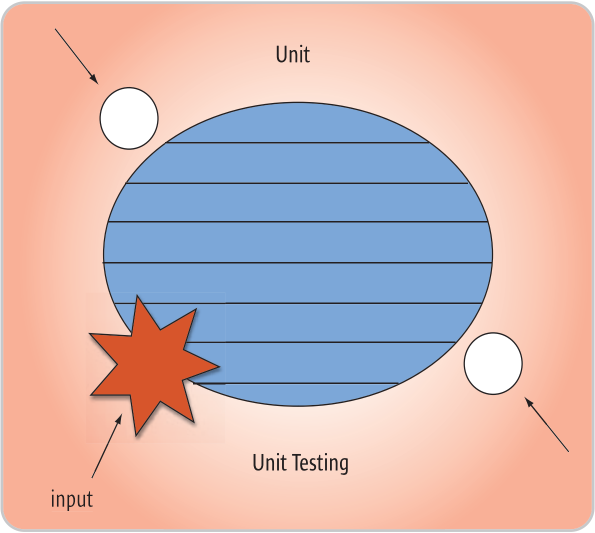


Figure 2 Unit testing

As Figure 2 illustrates, testing at the unit level offers a more effective way to find errors. When you test one unit apart from all other units, reaching potential errors is much easier because you are much closer to those errors. The difficulty of reaching the potential errors when the unit is tested independently is comparable to the difficulty of hitting one billiard ball into a particular pocket with a single stroke.

To expose the maximum number of errors that pose security threats, you need to exercise the code as fully as possible. Unit-level testing is the ideal mechanism for doing this.

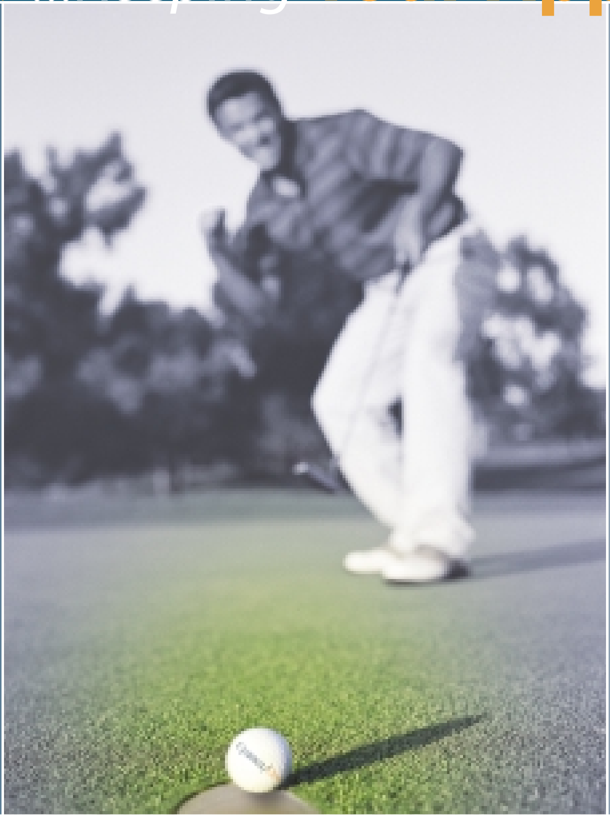
Conclusion

Following the strategies mentioned in this article will protect your Java code from many common security attacks. However, these strategies should be viewed as the beginning of a thorough security strategy, not as a security panacea. You need to anticipate and protect every possible security vulnerability because just one vulnerability can allow a skilled or lucky hacker to steal classified information or make your application unavailable. To ensure security, you need to learn how to identify and repair additional security vulnerabilities; some places to start are:

- *Compaq. Extended Static Checking for Java, SQL Injection Walkthrough:* www.securiteam.com/securityreviews/5DP0N1P76E.html
- Graff, M.G., and van Wyk, K.R. (2003). *Secure Coding Principles and Practices.* O'Reilly.
- Howard, M., and LeBlanc, D. (2002). *Writing Secure Code.* Microsoft Press.
- Kolawa, A., Hicken, W., and Dunlop, C. (2001). *Bulletproofing Web Applications.* John Wiley & Sons.
- McGraw, G., and Felten, E. (1999). *Securing Java.* John Wiley & Sons: www.securingjava.com
- Rao, R. (1999). *Writing Secure Java Code.* Macmillan Technical Publishing.
- *Sun Microsystems. Security Code Guidelines:* http://java.sun.com/security/seccodeguide.html
- Viega, J., McGraw, G., Mutdosch, T., and Felten, E. "Statically Scanning Java Code: Finding Security Vulnerabilities." IEEE Software (Sept./Oct. 2000).

Listing 1

```
void method (int a) {
        int b;
        int value_returned =  calculate (a, b);
         String user_name = getUserName( );
        boolean valid_stack_trace  =
isValidStackTrace(newThrowable().getStackTrace());
         assertTrue("Warning!, Unexpected sequence of operations"
, valid_strack_trace);
       // critical call, will not execute if assertion fails.
        updateDataBase(user_name, value_returned);

}
```



Joe Winchester
Desktop Java Editor

Fine Grains
Choke the Client

Recently I was having a discussion with a colleague about traditional versus Web clients. Instead of hearing the usual defense about how much easier it is to deploy and manage a thin client application, his point was that client/server fails because fine-grained transactions don't work.

With a browser-based application, when collecting data the typical workflow is the user inputting data on a number of pages and completing the transaction with a final Submit button. The flow of navigation is tightly controlled by the program that may create a session EJB to capture the data entered by the user. Since all the pages that users can enter their information on are known and controlled, the session bean has knowledge of all the data it needs to collect as part of the overall process. It also has a very obvious transaction boundary with the final "Submit" request where the changes become permanent.

By contrast, a traditional client application tends to have more freestyle navigation. Users are presented with tables or lists of queried information and from there they open additional detail screens to manipulate and enter information. The problem arises because on an edit screen the values of these objects can be changed. Having confirmed this by closing the window, does this mean that the update is confirmed to the database or just locally to the JVM to be committed as part of a larger transaction? If the value is captured just in-memory, then everyplace that same object is shown to the user should also be affected, including the original list that possibly shows the name in a table column. Any additional screens that might be used to reedit the same object or show its details should also be affected by the change. This means that the object must be uniquely identified within the Java program, so some kind of single instance management is required.

What if during the transaction the user performs another object query? The user could expect this to be a mixture of persistent data merged with any local changes he or she has made (as yet uncommitted). Including uncommitted newly created objects in the list is difficult as these must be woven between the persistent ones at the correct sort order positions. The problem is further compounded by the fact that within an overall transaction in freestyle GUI entry mode, the user might launch another multiscreen process to enter a new code and then decide not to commit this on the final screen. Alternatively, if the user does decide to commit this second process, it should be confirmed only into the world in which the original (still yet uncommitted) objects live. Nested transactions are required to support this, and apart from being very tricky to program, they are arguably confusing to the user as to which parts of the data are persistent and which aren't.

> "What is interesting about adopting a wizard or a rigid or modal dialog style of data entry is that this is essentially what a browser does"

The following are two solutions that can be employed.

First, use a persistence framework. Several good ones exist such as Cocobase, TopLink, or Solarmetric. If you want, you can always roll your own framework, although be warned it's a tough task to undertake. The danger is that you'll spend too many cycles building your killer framework and fall easily into the trap (as I once did in a previous life) of spending too long polishing your silver bullet while losing focus on the real business problem you're trying to solve.

The second solution is the one that I find that in my older years I tend to use more and more: solve a problem by avoiding it. The freestyle mode of client GUIs works well for navigation, but for data entry it keeps the user down a specific path. This is essentially what wizards do with their back and next buttons and a final "Finish" page, and it's obvious to the user that by not confirming, the final screen loses the whole process; and since it's modal there is no danger of having to merge objects into ones that are read freshly from the persistent back end.

What is interesting about adopting a wizard or a rigid or modal dialog style of data entry is that this is essentially what a browser does. Instead of the server dealing with atomic and essentially unpredictable requests for data updates, the scope of what is being changed is predetermined and known to the session. It is an easier programming model to deal with, and the fact that the browser does it well doesn't mean that traditional GUIs can't adopt it as well. There is still a large functional value that results from giving the user a client that can perform local validation and has the high usability function point that client windows offer; however, there is also a lot to be said for simplifying the input programming model to be based around a constrained workflow.

Several years ago I was with a customer who was building an entire client application framework in Swing that was being specifically designed to emulate a browser (complete with back and forward buttons). The customer's argument was that their users were comfortable with the browser. While I originally dismissed their project as dumbing down the power of what a proper GUI could do, I think there's a lot to be learned by looking at the mechanics of how a browser-based application does its persistence and using this session-based programming model for more traditional clients.

Resources

- *Cocobase:* www.thoughtinc.com
- *Solarmetric:* www.solarmetric.com

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

joewinchester@sys-con.com

June 28–
July 1, 2004

Moscone Center
San Francisco, CA

Join James Gosling, the father of the Java Programming Language

Java™

Everywhere starts here

Java™ technology is everywhere, improving the digital experience for everyone. It all starts at the JavaOne℠ conference, your source for cutting-edge knowledge and proven solutions. Discover from the experts how to deploy Web services and connect the world securely; you'll learn to code simpler and faster, and bring higher efficiency and profitability to your business.

The JavaOne conference offers hundreds of in-depth technical sessions in:

- The Foundations: Core J2SE™ Technologies
- Core Enterprise Technologies
- Java™ Technology on the Desktop
- Dissecting the Implementation: Solutions
- Intriguing and Unexpected: "New and Cool"
- Java Technology for the Web
- Java Technology for Mobility

JavaOne℠

Sun's 2004 Worldwide Java Developer Conference℠

Save $200! Register by May 31, 2004, and receive the Early-Bird price for the full Conference package. Registration code: ADARZKND

Register at java.sun.com/javaone/sf

Sponsored by

Sun microsystems

Produced by

MediaLive™ INTERNATIONAL

FEATURE

...and improving performance

ESCAPING SWING'S LIMITATIONS WHEN DRAWING GRAPHS

by David Shay and John Hutton

Swing is a general and flexible library for drawing graphical user interfaces. However, when trying to use Swing components to draw nodes in a graph, generality and flexibility have their limitations.

The Problem

We once worked on a project where we had to graphically display the configuration and topology of a telecommunications network. From the user interface programmer's point of view, this task is similar to drawing a graph, that is, nodes connected by links.

The user should be able to display as well as interact with the graph elements – move them, zoom in or out, display a pop-up menu, and so on. Our first thought was to use a JComponent as the base for the graph elements. Swing can take care of all the work we don't want to bother with, such as redrawing overlapping components and distributing the events to the correct graph element.

Since our nodes were nothing more than a label with an icon, we used the JLabel as a base class. Other Swing components made our job easy. A comment could be represented by a JTextArea, a table of attributes by a JTable, and links and other objects were simple JComponents in which we could override the paintComponent method.

This solution was quite easy and effective up to a couple hundred nodes. However, as the number of nodes and the complexity of the network grew, we quickly realized that Swing would not scale to our needs.

The best and most spectacular example to demonstrate Swing's lack of scalability is the moving of a node. If you have a small network of 20 nodes with a couple of links, you can grab a node, move it around, and it will follow the mouse pointer pretty smoothly. However, when you increase the number of nodes, you start to notice a latency time between the moment you initiate the move and the moment the node actually starts to move. When your network reaches the size of a real-life telecom network, the latency lasts over a couple of seconds. If you keep moving the mouse pointer, the node will try to follow it without ever managing to catch it.

David Shay is a software developer for Ericsson in Hungary. For the past five years he has worked at building GUIs in Java for telecom modelling tools.

david.shay@ericsson.com

John Hutton is as an information specialist for Horizon Technology.

john.hutton@thinkhcs.com

Solutions

Use the Memory You Really Need

It turns out that JComponents are so general, they contain a lot of variables and checks that are only overhead in our case. We decided to avoid using JComponents for our graph elements. In the same way that Swing uses the AWT Frame or Dialog as its top-level container and then takes care of everything else within the window, we use JComponent as our top-level component and take care of everything else within its bounds.

Stripping away everything that was unnecessary and keeping only what we really needed, we came up with objects optimized for our needs. We had the attributes we wanted to use, and the methods had less overhead because we had full control. This helped us counter one of the big problems of Swing: in spite of its flexibility, there are a lot of things you have no control over.

Table 1 compares the memory consumption of our graph elements (using JDK 1.4.1).

Most of the objects on a given display were made of links whose size was brought down to 256 bytes, less than a quarter of the size of its Swing counterpart.

Memory can be saved by offering interfaces only to the developer instead of an all-knowing base class. In this way, just the necessary information will be saved in the elements, and we get rid of any unwanted or unnecessary default behavior.

Repaint Only What You Need To

The area in which Swing quickly reaches its limit is the speed of the repaint. In our experience, Swing paints more graph elements than it should. Some objects get repainted even if their appearance was not modified, or if they were not overlapped by an object that triggers a repaint.

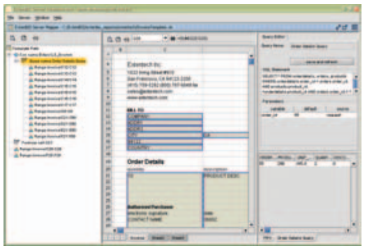
When component *X* needs repainting (its color has changed or it has changed location), Swing looks for other components that also need to be repainted. To do so, it iterates through all the siblings of *X* and checks if a sibling's bounds are intersecting with *X*'s bounds. If the bounds are intersecting, it repaints this sibling and also checks if it intersects with another one, creating a cascading effect of repaints. You might say that objects are rarely that close to each other, but we are drawing a graph of nodes connected with links. Link objects propagate the repaint to a lot more objects than you might think.

To cut short this cascading effect, simply avoid doing the intersection checks for the siblings. This will modify the *Z*-order of the elements in the graph, but this usually is an acceptable effect for the users.

The other reason why Swing repaints too much is the use of a rectangular shape as bounds for the components.

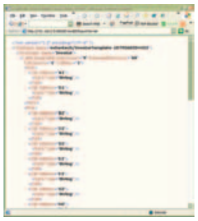
Apollo 17 Astronaut driving Lunar Roving Vehicle (LRV). December 1972

ExtenBIS: Know your business, accomplish your mission.™



ExtenBIS Data Mapper

XLS Output



XML Output

ExtenBIS Business Intelligence Server

Extentech understands that your users need their data presented with all of the capabilities of an Excel spreadsheet, but don't have the bandwidth to learn yet another report design tool. Well, what better tool to create advanced spreadsheet reports with than Excel itself?

With the **ExtenBIS Data Mapping** and **Deployment Wizard**, you can visually map your real-time database, session, and bean values to new or existing Excel templates with a drag-and-drop interface, then deploy it over the web to any server running the **ExtenBIS Servlet** with a single mouse-click. By creating server-based Business Intelligence applications from these files, ExtenBIS is the key to achieving the maximum ROI from your existing BI content.

Utilizing a fully-customizable JSP-based portal (source-code included) the ExtenBIS Business Intelligence Server delivers real-time Excel reports to your users anytime, anywhere.

EXTENBIS™
BUSINESS INTELLIGENCE SERVER

- ✓ Easy to use 3-Step Report Mapper and Deployment Wizard
- ✓ Secure Delivery of Reports via Web, Email, and File Drop
- ✓ Customizable Business Intelligence Portal with Source Code
- ✓ Customizable Business Intelligence Servlet with Source Code
- ✓ Extensible Object-Relational CellBinder API with Source Code
- ✓ XML Report Definition Format

Download ExtenBIS: www.extentech.com/jdj/

extenTECH™

Image AS17-147-22526 provided courtesy of NASA. NASA does not necessarily endorse Extentech Inc. products or service, or companies, trademarks, products or services herein mentioned.
Java is a Registered Trademark of Sun Microsystems Inc. Excel is a Registered Trademark of Microsoft Corporation. All other trademarks mentioned herein are the property of their respective owners.

Although most components you'd find in a user interface are roughly rectangular, when you look at links, you'll notice that nothing is less rectangular than a simple line. A lot of links get repainted because the intersection calculation takes into account the entire rectangle containing the line. It's quite easy for two rectangles to overlap, even if their diagonals do not.

If we are in control of everything, we can simply replace the rectangle with something else, let's say a shape or an array of rectangles. We can even ask the graph element if it intersects with a given rectangle, providing the developer with a higher level of flexibility. This complicates the intersection calculation, but the slow operation here is the repaint, and it's worth the extra effort if we can avoid several repaints.

An example of such a function in a Link element could look like this:

```
public boolean intersectWith(Rectangle r)
{
  return r.intersectsLine(fromX, fromY, toX, toY);
}
```

Control the Repaint Mechanism

The way Swing repaints is also an obstacle to optimization. A repaint invariably starts with drawing a rectangle the size of the component using the background color. This means that whatever is behind it will be erased and need repainting. In other words, whatever will be deleted will be added to the repaint queue, causing further deletion and repaints. We propose implementing the repaint as a two-step operation: deletion and drawing.

Instead of only one repaint queue, you can implement two queues: a deletion and a drawing. Graph elements can decide for themselves if they need to be deleted, redrawn, or both by adding themselves to one or both queues.

Not all types of repaint operations require deletion. For example, changing the color of a link needs only a repaint of the link, not of the objects it crosses. This will paint the link on top of the object it crosses, but again, modifying the *Z*-order of the graph elements has to be an acceptable side effect. Also, removing an element from the graph requires a deletion operation only.

The drawing algorithm deletes only those rectangles stored in the deletion queue, adding the objects they overlap to the drawing queue (provided they are not already there). Following that, the drawing queue is processed and all objects are painted only once. These queues can also check for duplicates, so that a deletion or a drawing needs to be processed only once.

The best optimization that Swing offers for limiting repainting is the clipping rectangle, but that also has a limit. If you know the extent of what needs to be repainted, you can give it as a parameter to the overloaded version of repaint. However, in most cases it's quite hard to know what exactly will be repainted. If a node is moved, several links and attribute tables may move too.

Swing Component	Size (bytes)	Graph Element	Size (bytes)
JComponent	490	DefaultObject	160
JLabel	706	DefaultNode	224
JTable	5027	AttributeTable	184
JGrLink*	1037	DefaultLink	256

* The JGrLink object is a home-made object deriving from JComponent.

Table 1 Memory consumption

It's not the job of the node to calculate the extent of the clipping rectangle. This feature is better implemented inside the Paint Manager, which already knows what it is going to repaint. Going through the deletion and drawing queue, it's easy to find out what really needs to be repainted. Once again, there's extra work in calculating the union of the bounds of all the graph elements, but the savings at the end is worth the game. In some cases, like zooming, when you know that everything will be drawn, you can simply bypass this check and tell the Paint Manager to draw everything without calculating the clipping rectangle.

Repaint Only When You Need To

Another automation of Swing that makes operations such as moving and zooming slower than they should be is the repaint on standard operations. Changing the size or location of a component provokes an immediate repaint. If you consider that zooming requires changing the location and size of all the elements in the graph, you have an idea of how slow this operation can be when each element repaints immediately upon its change.

Every call to repaint costs a lot and these should be kept to a minimum. Removing the Swing component from the chain allows you to reduce this behavior.

Another benefit is the location and size information, which is unnecessary for link objects. Links are just a line drawn between two nodes. Where is the need for location and size information? In the first version of our library, when a node was being moved, it triggered the following piece of code in the link element, which in turn triggered two repaint events:

```
Rectangle bounds = new Rectangle (from.getX(), from.getY(), 0, 0);
bounds.add(to.getX(), to.getY());
setLocation(bounds.x, bounds.y);
setSize(bounds.width, bounds.height);
```

Now this piece of code completely disappears. Even better, size and location can completely disappear from some elements such as the links. Throwing away the Swing components allows you to use absolute coordinates when drawing your links, instead of having to draw from (0,0) all the time. What good is this? Look at how simple the code for painting a link becomes:

```
public void paint(Graphics g)
{
  g.drawLine(from.getX(), from.getY(), to.getX(), to.getY());
}
```

Storing Your Elements

A Swing JComponent stores its children in a private array. There is absolutely no control over what is going on in this array, but sometimes you would like to be able to store these components in another type of structure adapted to your needs.

For example, when the user wants to select an element, it would be faster to look through the graph if it were stored in a quad tree. If you decouple the graph element container from the component, you could choose the container that suits your needs. You could handle ordering, apply filtering, or offer a different search criterion. Once again, you have complete control.

Other Optimizations

We'd like to finish this article with some optimizations that can be made on every graph drawing application, regardless of whether it uses Swing.

First, we mentioned several times the modification of the *Z*-order of the objects, which is a side effect of the drawing algorithm. Modifying the *Z*-order can actually be ugly – if your links come in front of your nodes, for example. If there are types of objects between which the ordering matters (like between nodes and links), it's possible to store your graph elements on different layers. Each layer will have its own drawing mechanism, a buffered image on which to draw, and a container of its own graph elements. It not only allows you to group like objects together, but you can also ensure that your links will always appear behind your nodes. Also, repainting a node will not affect the links, as each layer computes its dirty bounds separately.

For moving operations, an effective optimization is to add an extra layer, which I call the glass layer. When moving several graph elements, you can transfer them to the glass layer where they will be the only elements to be repainted during a move. When the move operation terminates, you can transfer the graph elements back to their original layer.

Transferring objects between layers can be a slow operation because it requires a deletion from the original layer. When moving an object, this deletion causes a delay before the graph element can actually be moved on the glass layer. One solution is to leave the elements on their original layer, while a copy of the element is being moved on the glass layer. When the move operation terminates, the deletion will occur on all layers, keeping the slow operation for the end of the move when the user is busy looking for his or her next action.

Finally, some graph elements can have quite complex algorithms to draw. Of course, it's possible to cache some of the values in the renderer so that the drawing happens faster. But the moving usually invalidates those caches, and the algorithm will have to be applied all the time. One solution is to replace the renderer on the fly with a simpler one during a move operation. For example, we had a link that was moving together with an arrow showing in which direction the information was flowing. Calculating the angle of the arrowheads is a complex operation. While moving, the renderer was replaced with another one, which represented the complex link as a simple black line.

Where to Go from Here

In this article we covered some of the improvements that can be made to Swing behavior to draw graphs more efficiently. While you can manage to trick Swing in several ways to get over some of these difficulties, in some cases you'll reach the limits.

Rewriting the painting and event distribution can be a painful and long operation, but it is worth the trip. It provides performance improvement as well as control over what is really going on under the hood.

UltraLightClient

by Canoo Engineering AG

Reviewed by
Peter Leitner

Remote Swing or server-side Swing – this is the most concise characterization of Canoo's UltraLightClient library (ULC). ULC offers server-side peer classes for Swing. For each Swing widget, there's a peer ULC class with essentially the same API.

The value added by ULC is the built-in split between client and server: ULC splits each widget into a client part and a faceless server part, and synchronizes these so-called half widgets at runtime.

The result is a client that's rich but thin, an idea that sounds puzzling today, since we associate rich clients with fat clients, and thin clients with HTML-based poor clients.

Minimal Footprint

An important characteristic of ULC is its minimal footprint. Despite the fact that it's a client/server technology, it imposes neither a framework nor an application server onto its user.

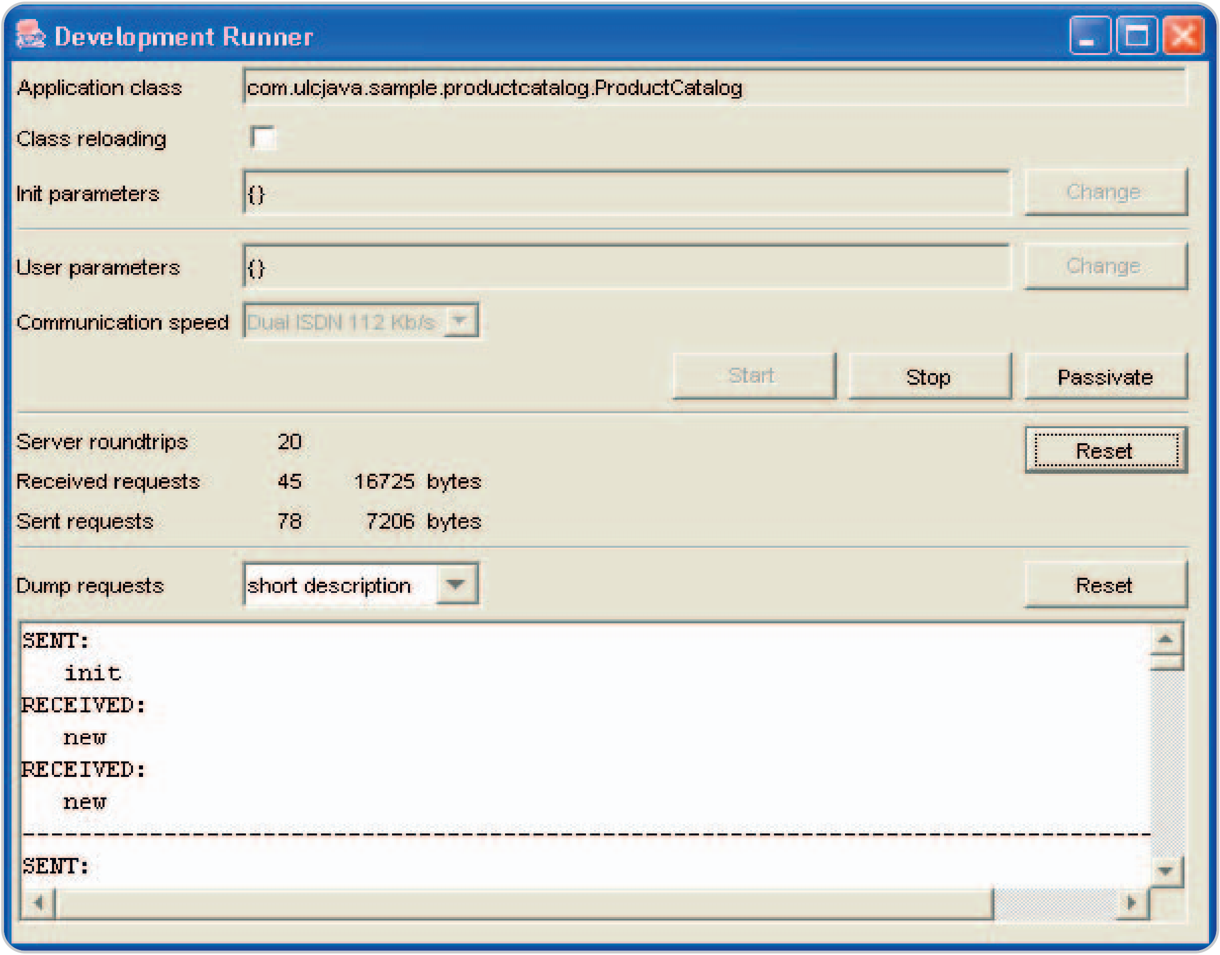
All infrastructural tasks are delegated to standard J2EE. The client relies on native Swing, communication is configurable as HTTP(S) or RMI over IIOP, and the server half widgets may run within a servlet or in an EJB container.

Conceptually, ULC is just a smart widget set. Its impact on an application is limited to the presentation layer. Programmers can employ their technology of choice for business objects, persistence, and other software layers. The only constraint ULC imposes is the thin client architecture.

Rich Thin Clients

ULC seeks to combine the benefits of HTML thin client applications and fat client productivity applications. Its starting point is the J2EE architecture with a server-side programming and execution model.

This makes ULC applications conceptually similar to HTML-based J2EE applications: all the business logic and the model of the presentation logic execute on the server. The client is a gener-



Application class com.ulcjava.sample.productcatalog.ProductCatalog

Class reloading □

Init parameters {}

User parameters {}

Communication speed Dual ISDN 112 Kb/s

Server roundtrips	20	
Received requests	45	16725 bytes
Sent requests	78	7206 bytes

Dump requests short description

```
SENT:
   init
RECEIVED:
   new
RECEIVED:
   new
------------------------------------------------------------------------------
SENT:
```

Figure 1 DevelopmentRunner GUI

Peter Leitner is a team leader and senior software engineer at Würth Phoenix. He has 10 years of experience developing object-oriented ERP systems and online/offline sales support systems.

peter.leitner@wuerth-phoenix.com

ic presentation engine, like a browser. The difference between a browser and the ULC engine is that the latter handles descriptions of rich graphical user interfaces instead of HTML.

While not new, the concept of rich thin clients has lost none of its appeal. It promises state-of-the-art usability combined with easy manageability and operation.

Let's see to what extent Canoo's ULC lives up to this promise.

Getting Started

Canoo's offering includes:

- The core ULC library, currently v. 5.1.3
- A visual editor for the WebSphere Studio Application Developer IDE, currently v. 1.0 (an Eclipse version is scheduled for Q2004)
- A load and performance testing tool, currently v. 2.2

Installation is simple: you can either extract the distribution archive or use the platform-specific installer for Windows, Linux, Mac OS X, or Solaris. The distribution contains the ULC libraries for the server container and the presentation engine, a 250-page developer guide, a client/server simulator called DevelopmentRunner that enables executing applications in a single virtual machine, and the source code for six sample applications.

The best way to start is to run the sample applications, some of which are available as online demos on the Web site. To execute them on your own machinery, either double-click their launch scripts or drop one of the WAR Files into your preferred application server and invoke their start page within your Web browser.

The ULC-Set example is a good point of departure. It provides an overview of all widgets available. Each widget sample is a click away from its source code.

Given the rich set of examples, the familiar Swing API and the comprehensive developer guide, a Java engineer will learn quickly. My experience is that a programmer who knows Swing and the basics of client/server computing will be fully productive after a few days.

IDE Integration

A nice feature of ULC is that it fits into your preferred IDE. The only additional tool you need is the DevelopmentRunner that comes with the library.

The DevelopmentRunner executes the client and the server in a single virtual machine, shortening the edit-compile-test cycle. Using this tool is simple: just call it in the main method of your application class:

```
public static void main(String[] args) {
DevelopmentRunner.setApplicationClass(<Your
ULC Main Class>);
DevelopmentRunner.main(args);
}
```

Since you can execute a ULC application entirely within your IDE, debugging and testing is easy. Your IDE's debugger and your favorite tools will all work.

For the purpose of monitoring interaction between client and server, the DevelopmentRunner offers a dialog window:

```
DevelopmentRunner.setUseGui(true);
```

This window displays the messages exchanged and allows simulating different bandwidths (see Figure 1). I like this latter option because it enables me to test the real-life behavior of my application within my IDE.

If your favorite IDE is Eclipse or WebSphere Application Developer, use the drag-and-drop visual editor that comes as an add-on product (see Figure 2). This editor corresponds almost exactly to the Swing-based editors available for these platforms. It generates Java and reflects code changes back to the user interface.

The somewhat privileged positioning of Eclipse and WebSphere is also documented by the fact that the DevelopmentRunner's user interface is integrated in their workbench.

Working with ULC

We've been working with ULC for two years now and have developed two applications. One of them is WÜRTHPHOENIX CIS (www.wuerth-phoenix.com), a company information system that is shown in Figure 3. WÜRTHPHOENIX CIS supports profit centers in their financial reporting, planning, and projection. It was created by three developers within 18 months, and has been rolled out as a standalone application and in a hosting datacenter where it's serving over 280 companies worldwide.

The second application we developed is WÜRTHPHOENIX ERP-Basic, an enterprise resource planning (ERP) application for small and medium-sized companies. WÜRTHPHOENIX ERP-Basic

Canoo Engineering AG

Kirschgartenstrasse 7
CH-4051 Basel, Switzerland
Phone: +41 61-228-9444
Web: www.canoo.com

Specifications

Platform: Any platform that supports the Java Runtime Environment 1.2.2 or later
JVM: Java Runtime Environment 1.2.2 or later
Java Application Server: Any J2EE compliant servlet and EJB container

Pricing:

- **ULC Library:** $1,495 per developer. Includes an unlimited runtime license and all configuration options (deployment in servlet, EJB container, or standalone, with HTTP(S) or RMI/IIOP).
- **Visual Editor:** $499
- **ULC Load:** $3,000

Test Platform

Sun JRE 1.2.2/1.3.1/1.4.1/1.4.2.
Windows 2000/XP, Red Hat Linux, and Solaris.
BEA WebLogic, IBM WebSphere Application Server, JBoss, and Tomcat

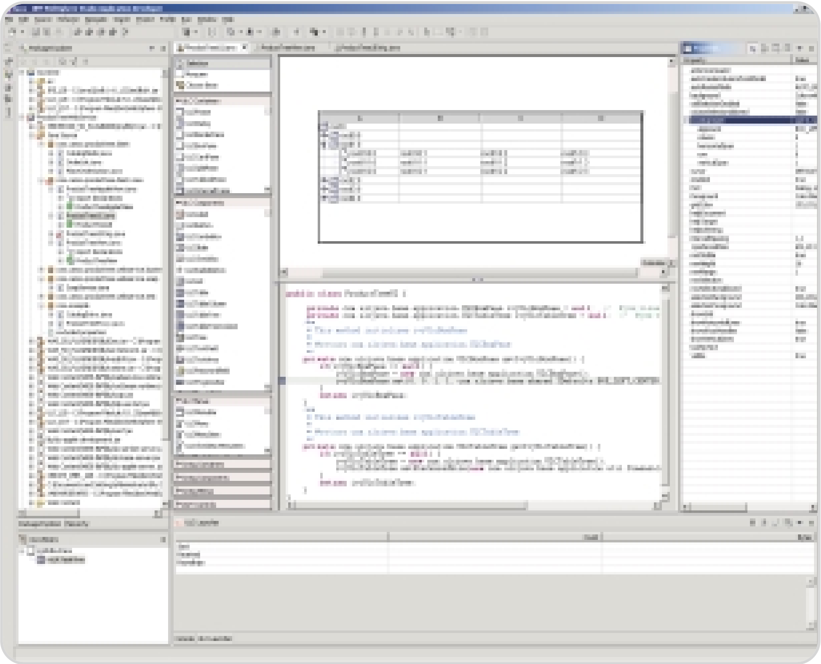


Figure 2 ULC Visual Editor

supports purchasing, planning, order management, and logistics, including inventory accounting as well as reporting and statistics. Ten developers completed it within 14 months.

For development, we use PCs with Windows 2000, running Eclipse 2.2.1, JDK 1.3.1, and ULC's DevelopmentRunner. An integration server and testing server round off our setup. They're running Windows 2000 and Linux Red Hat AS 2.1, respectively, with Jakarta Tomcat 4.1.29.

Our experience is that developing with ULC is indeed similar to working with Swing. The benefit as compared to Swing is that you essentially get a J2EE-compliant client/server application for free. The only client/server-related issue you need to remember is that your application runs in a multiuser and multithreaded environment:

1. Avoid using static variables, because they're not thread-safe. Instead, manage objects in your server session, like in HTML/servlet applications.
2. There is the option to communicate between sessions. You can, for example, realize a notification mechanism that propagates data changes to all sessions.
3. When opening a modal dialog window, the thread won't wait as in Swing. As a consequence, set up a listener that reacts to the action closing the dialog.

A positive surprise for us was the scalability of ULC's thin client solution: the built-in minimization of client/server communication and the economic use of memory on the server work well for us. They enable our users to work over a worldwide VPN or over the Internet, sometimes connecting with slow modem lines, and to use a cost-effective server infrastructure.

A further important point for us is manageability of releases and production. Our applications run on a variety of platforms, for different customers. On the client side, the 350K presentation engine runs under Windows NT, 2000, and XP as an applet and a standalone Java application distributed with JNLP. On the server side, the applications run on Windows 2000, NT, XP, and Linux AS 2.1. ULC's J2EE compliance and the fact that it runs on any JDK from 1.2 upward enable us to support this wide variety with a single code base.

Extending ULC

It's possible to extend ULC to support new kinds of widgets and data objects. Example extensions are special value formatters for text fields, specific enabling strategies that are needed on the client, or widgets supporting graphic functions. There's an extension API that we've used to develop a special table widget. Unfortunately, the code samples of the distribution contain only one such extension. I wish that Canoo included more of them, or, better, offered a community service for the exchange of add-ons and code snippets.

Performance

ULC employs a number of techniques to minimize round-trips and communication, including local validation, lazy loading, caching, and compression of messages. Yet, the performance of an application never depends on its presentation layer alone. What counts is end-to-end performance, and this has to be measured individually for each application.

Canoo offers a tool for end-to-end testing, called ULC Load. With this tool you can record user interactions and replay scenarios in parallel to simulate any number of users. You can then measure response time and bandwidth. Furthermore, you can export the results for analysis. We found this tool to be simple and useful. It allowed us to test efficiently, and helped in the sizing of the production environment. Regarding performance, in our experience ULC was never the bottleneck in an end-to-end test run.

Summary

ULC is a lean but effective technology: you develop as if Swing widgets were running on the server. You have a homogeneous, server-side programming model, and get a scalable client/server application you can deploy in any J2EE environment.

This is doubtlessly one of the most efficient ways to realize rich client business applications, in particular if you can share the J2EE infrastructure with HTML-based applications, thereby eliminating the need for developing a separate solution for functions like security, logging, load balancing, or monitoring.

There are, of course, limitations. If an application needs more than the standard widgets offered by Swing, there is no out-of-the-box solution. For such cases ULC needs to be extended, typically by integrating third-party widgets or components.

Snapshot

Target Audience: Software developers
Level: Intermediate to advanced
Pros:

- Simplifies rich client development for client/server apps
- Small footprint, optimized client/server communication
- Deploys easily in many configurations
- Standards-based (J2EE, J2SE)
- Integrates with existing Web infrastructure
- Attractive pricing; no runtime license fees

Cons:

- Few out-of-the-box widgets beyond those of Swing
- Use of third-party graphics libraries possible but requires integration

JDJ Ranks No. 1 in the World in Digital Magazine Circulation Among All Magazine Titles

(*Montvale, NJ*) – SYS-CON Media, the world's leading i-technology magazine publisher, has announced that its flagship print magazine, *JDJ*, ranked number 1 in the world in digital circulation delivery. *JDJ*'s most recent six-month average circulation was 162,000 copies, of which 43% was requested by its subscribers to be delivered digitally. *JDJ*'s monthly digital edition is an identical replication of its print edition, and SYS-CON's advertising partners enjoy an incredible bonus exposure beyond its 60,000 rate base, which has not been increased since *JDJ*'s first issue.

Circulation Management Magazine reported on March 9, 2004, that "*JDJ* is leading the way with 69,776 digital only subscriptions, or 43.1 percent of its total qualified circulation of 162,091, according to an analysis by *The Circulator*" (http://circman.com/ar/marketing_java_developers_journal/index.htm).

According to an M10 Report article written by Ted Bahr, "The top ten digitally delivered magazines are all tech titles, with *JDJ*, published by SYS-CON Media, leading the pack with 43 percent of its file delivered digitally" (www.m10report.com/html/227.asp).

M10 Report ranks the top 10 magazines as follows:

JDJ	162,019	69,776	43.1%
eWeek	422,600	57,500	13.6
NASA Tech Briefs	195,405	23,948	12.3
Network Magazine	200,000	12,788	6.4
SD Times	51,443	11,879	23.1
Design News	170,243	11,487	6.7
Microsoft Certified Professional	111,834	9,933	8.9
EE Times	153,823	6,323	4.1
Wireless Week	31,730	4,627	14.6
Control	70,050	3,967	5.7

SOURCE: BPA International publisher's statements. NOTE: Under BPA International rules, a subscriber who has opted to receive both the print and digital editions of a publication is reported only once and counted only once within that and the corresponding % of total qualified circulation of those combined subs are provided only for informational purposes. 32,320 *JDJ* subscribers receive the magazine both in print and in its digital edition, but counted as only one within the magazine's qualified circulation numbers.

Teamstudio Releases Edition 5 of Java Tools Suite

(*Beverly, MA*) – Teamstudio, a provider of agile software tools for application developers, has announced the release of Edition 5 of its Teamstudio for Java suite, which introduces a new Java development tool, Teamstudio Thread Profiler. It also delivers new enhancements to existing tools in the suite: Teamstudio Analyzer for Java, Teamstudio Memory Profiler, and Teamstudio Performance Profiler.

www.teamstudio.com

LISA 2.0 Automated Testing

(*Dallas*) – iTKO, Inc., an enterprise software development tool provider, has announced a testing breakthrough to ensure that companies can deploy Web services with confidence. LISA can uncover and report the exact source of any Web service problems stemming from Web and app servers, network transmission, database storage and retrieval, middleware, other applications, and attached components all the way down to the most granular Java and XML code level.

www.itko.com

RetroVue **1.1**

by VisiComp Inc.

Reviewed by
Klaus Berg

It's unnecessary but true: a lot of Java programmers still debug by putting System.out.println() statements in their code to find out what the program is really doing and where the problems are. To overcome this antiquated approach I've tried several debuggers: Sun's JDB is free but cumbersome and hard to work with. Visual SlickEdit, my favorite IDE, has integrated debugger support but is also difficult to run. Metamata debugger was my next choice but the product went away. So I searched the Web for options and, indeed, there is a better way: RetroVue from VisiComp. It creates a complete journal of your program's execution, letting you go back to any previous instance and examine the state back then. These features and some articles about "Omniscient Debugging" aroused my interest in RetroVue.

In this article I try to explain the difference between RetroVue and other debuggers. Then you, the reader, can decide if it's worth paying $995 for it.

Omniscient Debugging

RetroVue is built on the concept of Omniscient Debugging. The idea is to collect operations at each "point of interest" (setting a value, making a method call, acquiring/getting a lock, throwing/catching an exception) in a program and then allow the programmer to use those events to explore the history of the program. A free (GPL) experimental debugger called ODB (written by Bill Lewis) is an implementation of this idea (www.lambdacs.com/debugger/debugger.html).

VisiComp Inc.'s software analyzer RetroVue 1.1 is a full-fledged product, also written in Java, with a strong graphical user interface based on Sun's Swing library. It works by inserting byte codes into the application's class files. This code collects information on each point of interest as mentioned. Collected events or operations are stored in a log (RetroVue calls it a journal file) and then displayed in GUI. A programmer can use the GUI to review the behavior of objects, variables, and method calls. This means that you see which values are bad, find out where those values came from, and who set them and why. This also means that there are no nondeterministic problems. You don't have to guess where the problems might be; you don't have to set breakpoints; you don't have to wonder which threads ran when; and you don't ever have to repeat a program run.

Where Am I? – Breakpoint Debuggers and Flight Recording

With traditional debuggers the programmer generally only knows that the program is stopped at a given breakpoint, but not the states leading up to the break. Omniscient Debugging can solve this problem. RetroVue is characterized by VisiComp as a "Total Recall Debugger" for Java developers. This hints at where the tool is positioned in the software life cycle: early in the development phase. There are other "flight-recorder" tools like RootCause (www.ocsystems.com/prod_rootcause.html), designed to simplify tracing and data collection in a post-development environment, i.e., in the production environment. Once an application transitions to testing, integration, or QA, development tools are no longer appropriate. They place too much load on the system. That's true for RetroVue because RetroVue logs and gathers data from nearly all aspects of the Java app's execution. There is a lot of similarity in the underlying technology but RetroVue is intended to be a developer's tool, to be executed in the development environment. The price the developer has to pay is performance, i.e., RetroVue slows down execution and perhaps needs an additional disk for its journal file. This is because RetroVue works on a disk-based model rather than a memory-based one. The current version of the product only loads what it needs into memory, chucking what's no longer necessary to make room for stuff that becomes needed as the user examines further events. However, the bigger the journal file, the larger the amount of information that's required. RetroVue can work with huge journal files where the actual limit depends on the particulars of the journal file contents. Of course, if the size of the working set exceeds the actual RAM in your machine and the underlying virtual memory subsystem is heavily utilized, performance will drop off rapidly. Thus RAM does really matter and I recommend having at least 512MB.

VisiComp, Inc.

6630 Highway 9
Suite 103
Felton, CA 95018
Web: www.visicomp.com
Phone: 831 335-1820
Fax: 831 335-7038

Specifications

Platforms: Windows, Unix, Linux, Mac with JDK 1.3+
Pricing: $995 per developer

Test Platform

Fujitsu-Siemens PC Scenic W600 with Intel Pentium 4 3,06GHz processor with 1GB RAM. Windows XP Professional SP 1.

Snapshot

Target Audience: Java programmers and developers
Level: Beginner to advanced
Pros:
- Easy installation
- Powerful and intuitive GUI
- Easy to learn

Cons:
- Not targeted at server applications
- No filter capabilities
- No instrumentation of Java classes

Klaus Berg is a principal engineer with Siemens AG, Munich, Germany. He has worked for years as an architect and implementor in Java GUI development projects with Swing and Java Web Start. Now he develops server-side J2EE intranet applications but from time to time he comes back to the Java desktop.

klaus-peter.berg@siemens.com

Facts and Features

As mentioned before, RetroVue instruments your Java byte code, either on the fly as the program is running or offline from the command line. Unlike conventional debuggers, RetroVue keeps track of every operation executed by your program. By maintaining a complete journal of every assignment to every variable, each method invocation and return, each thrown and/or caught exception, each thread switch, each lock operation (e.g., when executing synchronized methods, etc.), RetroVue allows you to scroll forward and backward in time and lets you examine the state of your program at any given instant. Its powerful and intuitive Swing GUI lets you scroll time as easily as you scroll text in an editor. The GUI is based on VisiComp's visualization software, which uses proprietary algorithms to show the mechanics of Java programs in a graphical format that displays the state, execution, and thread interactions of programs. The main screen is divided into different optional views. I used RetroVue on two different machines, one with a 15" monitor. My experience is that an 18" or 19" monitor is better suited for making all the interesting views visible at one time without the need for scrolling or resizing some panes.

The following panes are available (some are shown in Figure 1).

- ***Class Browser:*** To view and navigate the class hierarchy of your program. Only those classes currently loaded are accessible so you can't waste time looking for a bug in unused code.
- ***Search Results:*** Displays the results of search requests performed via a popup menu accessible in various other views.
- ***History View:*** Displays a "log" of every operation produced by the program being debugged. Change the variable historyview.showSequence Num=true in RetroVue's properties file (contained in retro.jar) to display event or operation numbers too. Perhaps this will be possible via the GUI Option menu in the future.
- ***Data View:*** The data for the current or selected method invocation frame. If the program being debugged was compiled with "-g", then the local variable names will be shown. If "-g" was not used to compile the program, the local variables' slot numbers will be used.

  When faced with a null pointer exception, RetroVue's "Show Previous Assignment" command in the data view makes it trivial to go back to the exact point where the variable was assigned a null value. By hopping backward in this way, it's easy to get from the manifestation of the bug (the symptom) back to when and where the problem actually occurred (the cause).
- ***Messages:*** Displays output, if any, from the program currently being debugged by RetroVue.
- ***Source View:*** Provides the text of the source files for your program. Source views will only appear if the Source Path preference was set when the program was launched under RetroVue. The Source View consists of different source viewers, one viewer for each source file being viewed. Each viewer is divided into two portions: the gutter and the text area. The user may set a breakpoint next to a line of code by clicking in the gutter. This breakpoint is a stopping point when navigating through the history view (see below).
- ***Thread View:*** Displays the chronological sequence of the information produced by your program, organized by thread. It provides a "high-altitude" overview of your program's information, rather than the detailed information provided by the History View. In the timeline at the top of this view, the numbers correspond to the same event ID numbers shown in the History view (when you have set the property historyview.showSequence Num=true).

  The colored bars show the activity within a particular thread and certain interactions between threads. Multiple threads contending for more than one lock can lead to a programming bug known as deadlock, which is more technically defined as a "cyclic lock dependency." RetroVue automatically detects this condition and alerts you with a skull and crossbones icon. Figure 2 gives you an impression of the Thread View.

Navigation Through Time

In addition to finding bugs, RetroVue lets you validate the expected runtime behavior of your program. You no longer have to insert println statements or set breakpoints to verify that your program is doing what it is supposed to do. Instead navigate through the History View using commands like "Play," "Step over," "Step out," "Step into," or "Rewind". A rewinding facility like this has never been an issue in traditional "breakpoint" debuggers, so you have to learn a new "methodology" of debugging.

Installation and Getting Started

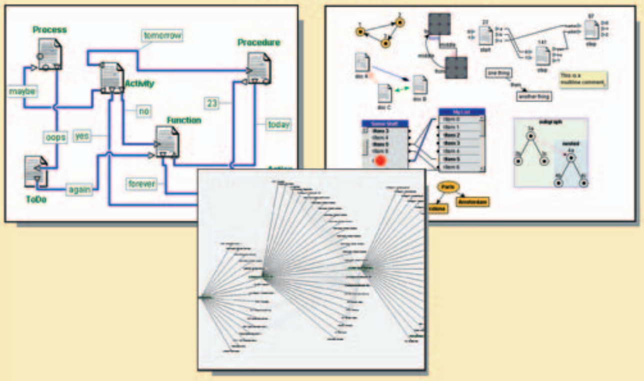
Installation of RetroVue is just a matter of seconds: download the appropriate installation file for your platform (for Windows, Unix, Linux, or Mac). Uncompress the file with a decompression utility like Winzip or tar in a directory of your choice. That's it. You can then modify your program's start statement and RetroVue automatically comes up if your program exits (under normal and exception conditions; but if you cancel your program using CTRL+C or something similar you have to bring up RetroVue from the command line).

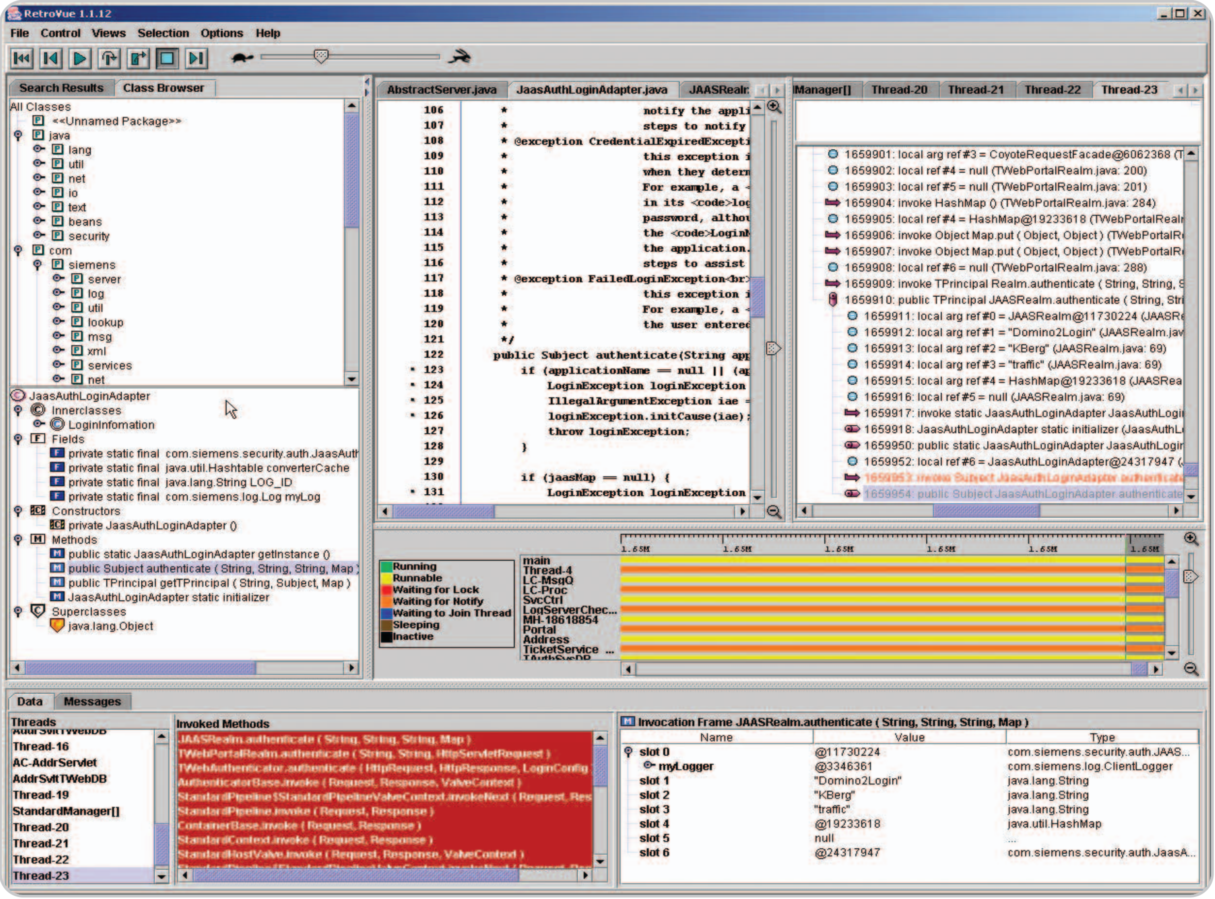
Modify your run statement as follows:

```
java -jar <retro-dir>\retro.jar -journal <journal-
file> -sourcepath <source directory> -exec -cp
<classpath> -D<properties> <your-main-class> <pro-
gram arguments>
```

Besides the command-line launcher, RetroVue comes with a GUI launcher that serves as a front end to RetroVue's command-line options.

The documentation is provided as an HTML manual. It's enough for getting started, but I would prefer an additional "How to" section that describes some typical procedures when using RetroVue for different debugging problems.





```
All Classes
  <<Unnamed Package>>
  java
    lang
    util
    net
    io
    text
    beans
    security
  com
    siemens
      server
      log
      util
      lookup
      msg
      xml
      services
      net
```

```
JaasAuthLoginAdapter
  Innerclasses
    LoginInfomation
  Fields
    private static final  com.siemens.security.auth.JaasAuth
    private static final  java.util.Hashtable converterCache
    private static final  java.lang.String LOG_ID
    private static final  com.siemens.log.Log myLog
  Constructors
    private JaasAuthLoginAdapter ()
  Methods
    public static JaasAuthLoginAdapter getInstance ()
    public Subject authenticate ( String, String, String, Map )
    public TPrincipal getTPrincipal ( String, Subject, Map )
    JaasAuthLoginAdapter static initializer
  Superclasses
    java.lang.Object
```

```
         *                         notify the appli
         *                         steps to notify
         * @exception CredentialExpiredExcepti
         *                         this exception i
         *                         when they determ
         *                         For example, a <
         *                         in its <code>log
         *                         password, althou
         *                         the <code>LoginM
         *                         the application.
         *                         steps to assist
         * @exception FailedLoginException<br>
         *                         this exception i
         *                         For example, a <
         *                         the user entered
         */
       public Subject authenticate(String app
           if (applicationName == null || (ap
               LoginException loginException
               IllegalArgumentException iae =
               loginException.initCause(iae);
               throw loginException;
           }

           if (jaasMap == null) {
               LoginException loginException
```

```
1659901: local arg ref #3 = CoyoteRequestFacade@6062368 (T
1659902: local ref #4 = null (TWebPortalRealm.java: 200)
1659903: local ref #5 = null (TWebPortalRealm.java: 201)
1659904: invoke HashMap () (TWebPortalRealm.java: 284)
1659905: local ref #4 = HashMap@19233618 (TWebPortalReal
1659906: invoke Object Map.put ( Object, Object ) (TWebPortalR
1659907: invoke Object Map.put ( Object, Object ) (TWebPortalR
1659908: local ref #6 = null (TWebPortalRealm.java: 288)
1659909: invoke TPrincipal Realm.authenticate ( String, String, S
1659910: public TPrincipal JAASRealm.authenticate ( String, Stri
  1659911: local arg ref #0 = JAASRealm@11730224 (JAASRe
  1659912: local arg ref #1 = "Domino2Login" (JAASRealm.jav
  1659913: local arg ref #2 = "KBerg" (JAASRealm.java: 69)
  1659914: local arg ref #3 = "traffic" (JAASRealm.java: 69)
  1659915: local arg ref #4 = HashMap@19233618 (JAASRea
  1659916: local ref #5 = null (JAASRealm.java: 69)
  1659917: invoke static JaasAuthLoginAdapter JaasAuthLogi
  1659918: JaasAuthLoginAdapter static initializer (JaasAuthL
  1659950: public static JaasAuthLoginAdapter JaasAuthLogin
  1659952: local ref #6 = JaasAuthLoginAdapter@24317947 (
  1659953: invoke Subject JaasAuthLoginAdapter.authenticate
  1659954: public Subject JaasAuthLoginAdapter.authenticate
```

```
Threads
AddrSvltTWebDB
Thread-16
AC-AddrServlet
AddrSvltTWebDB
Thread-19
StandardManager[]
Thread-20
Thread-21
Thread-22
Thread-23
```

```
Invoked Methods
JAASRealm.authenticate ( String, String, String, Map )
TWebPortalRealm.authenticate ( String, String, HttpServletRequest )
TWebAuthenticator.authenticate ( HttpRequest, HttpResponse, LoginConfig
AuthenticatorBase.invoke ( Request, Response, ValveContext )
StandardPipeline$StandardPipelineValveContext.invokeNext ( Request, Res
StandardPipeline.invoke ( Request, Response )
ContainerBase.invoke ( Request, Response )
StandardContext.invoke ( Request, Response )
StandardHostValve.invoke ( Request, Response, ValveContext )
```

Invocation Frame JAASRealm.authenticate (String, String, String, Map)

Name	Value	Type
slot 0	@11730224	com.siemens.security.auth.JAAS...
myLogger	@3346361	com.siemens.log.ClientLogger
slot 1	"Domino2Login"	java.lang.String
slot 2	"KBerg"	java.lang.String
slot 3	"traffic"	java.lang.String
slot 4	@19233618	java.util.HashMap
slot 5	null	...
slot 6	@24317947	com.siemens.security.auth.JaasA...

Figure 1 RetroVue GUI with different panes

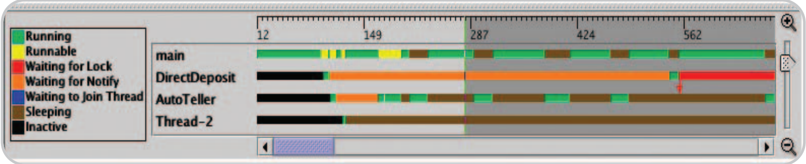


Figure 2 RetroVue's Thread View

Current Limitations and Future Enhancements

The current version, RetroVue 1.1.12, is not really intended for Java enterprise applications (RetroVue Enterprise Edition is on their schedule), but there are situations when you can use it even there. If you examine the run statement above you'll see that you can analyze part of your J2EE Web app if it's started as a process in its own JVM! Version 1.1.12 of RetroVue can only be used for postmortem debugging and analysis of an already-executed program. Future versions are being planned to allow interactive debugging of a running program.

Another limitation is that Java system classes like java.util.Properties are not augmented or tracked. This will be addressed in a subsequent release.

What I miss the most is a filter capability and a "Start/Stop-Recording" facility as in CPU-profiling tools. Adding another feature would be quite easy: changing the Swing GUI's look and feel. I'm not happy with the current Sun Metal L&F and I can imagine other users would also prefer having another choice. VisiComp's support team told me that all these wishes were on their list for future releases.

Summary

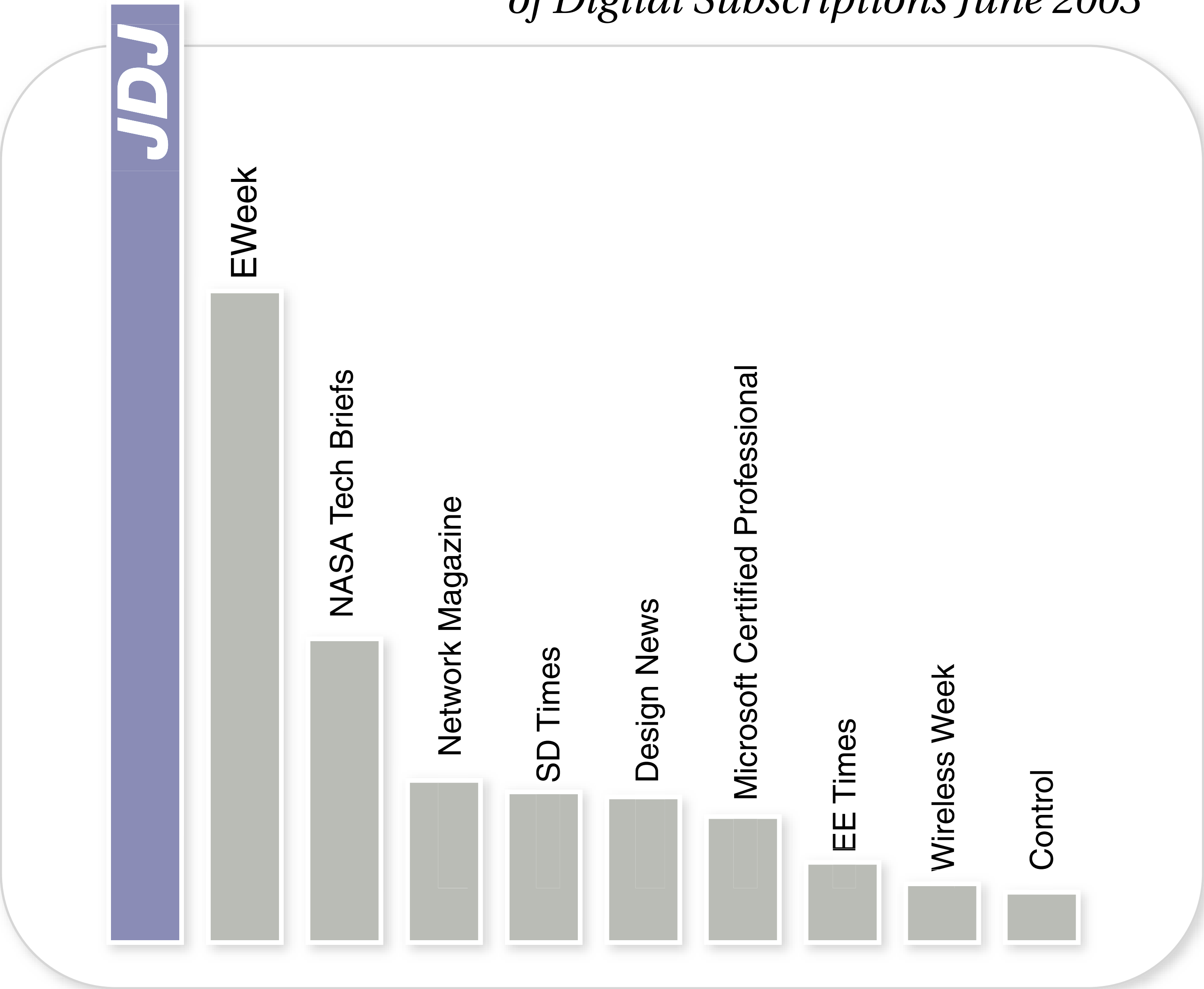
VisiComp's RetroVue debugger was featured in a James Gosling JavaOne 2002 keynote. Sun Microsystems vice president, distinguished engineer, and creator of Java said: "The work the folks at VisiComp have done is truly spectacular. It has a strong opportunity to dramatically improve the software development process."

I think if you are looking for a new debugging concept you should spend the time and money and try VisiComp's RetroVue debugger. If you're not fully satisfied – they give you a 30-day money-back guarantee (for certified users there is an evaluation copy available, too). Using RetroVue I was able to fix some horrible bugs in our J2EE Web app that uses an embedded Tomcat Server with different JVMs, each running different threads. I had to analyze a 38MB RetroVue journal file, but there were no problems with the program performance or the "speed" when loading the log file and working with RetroVue's GUI. However, I used a high-end PC running Windows XP with 3.1GHz CPU, lots of disk space, and 1GB of memory. As with a lot of tools: the more memory you have the better for RetroVue, but you don't have to adjust your java -Xms –Xmx options; this is done by RetroVue behind the scenes. On my home PC running Windows 2000 with 800MHz and 256MB memory, things looked a little bit different. It took about four minutes to load the 38MB journal file, and navigating in the GUI was significantly slower than on my high-end PC. The reason is simple: the total memory consumption was more than 500MB, although RetroVue's java.exe took only 48MB! Nevertheless, if you plan to use RetroVue with large log files, consider investing in at least 512MB memory. But having done so, debugging is fun!

Oops, we did it again!

JDJ No. 1 in the World

Top 10 Publications Ranked by Volume of Digital Subscriptions June 2003



SYS-CON MEDIA

Source: BPA International publisher's statements
Note: Under BPA International rules, a subscriber who has opted to receive both the print and digital editions of a publication is reported only once and counted only once within that and the corresponding % of total qualified circulation of those combined subs are provided only for informational purposes.

Miles Silverman
VP Marketing

Carmen Gonzalez
Senior VP Marketing

Contact | Carmen Gonzalez: carmen@sys-con.com or 201 802.3021

JAVA DEVELOPER'S JOURNAL

From Within the Java Community Process Program

Onno Kluyt

The 'groovy' JSR

Welcome to the April edition of the JCP column! Each month you can read about the Java Community Process: newly submitted JSRs, new draft specs, Java APIs that were finalized, and other news from the JCP. In this month's column I'm focusing mostly on one new JSR.

Java Is a Platform

The above is an often-heard description when talking about this technology. While many Java developers may first think of the Java programming language, what makes this technology successful is not only the niceties of the programming language but the richness of the class libraries and the presence of a virtual machine that can run anything as long as "anything" consists of valid bytecodes: the platform aspect. The virtual machine doesn't worry much about how those bytecodes that it's asked to execute came about. From very early on in the life of the Java technology, there have been efforts to run programs written in other programming languages in the Java runtime environment.

Nowadays most Ada programs are compiled to Java bytecode. You can compile COBOL to run on a Java runtime environment if you so wish. More recently, there have been other efforts related to new languages such as Ruby and Python where developers have created Java versions, respectively, JRuby and Jython. The JCP recognized the use of scripting languages in the J2EE technology environment last summer and is currently working on a JSR to provide scripting languages, such as PHP access to Java objects. This is JSR 223. Now, there is a new JSR that explores an altogether new angle.

A JSR for the Groovy Programming Language

Submitted by Geir Magnusson and Richard Monson-Haefel, and to be led by James Strachan and Richard, JSR 241 is on the JSR Review Ballot. The goal for this JSR is to standardize the specification for the Groovy programming language so that platform implementers, tool vendors, and others can provide compliant implementations for their developers to use with the Java platform. Groovy is based on J2SE 1.4. In the words of the submitters: "Groovy is a complement to the Java programming language. Where the Java programming language is exacting, Groovy is expedient. Where the Java programming language is extensive, Groovy is convenient."

I asked James Strachan via e-mail how Groovy came to be. James explained that last summer he observed that several Java developers were intrigued by features found in new languages like Ruby and Python. These are great scripting languages to write the glue that holds enterprise applications together. While they are very useful, James felt that what was missing was a language that provides access to the J2SE and J2EE APIs for this purpose. That is the API set that the developers are familiar with, and Java developers will want to retain their investment in learning these APIs and the development tools that support them. And so Groovy was born. It began as an experiment to see whether a Ruby-like language could be compiled to bytecodes and use the Java runtime environment without wrapping Java objects but it quickly gained a lot of momentum, and now Groovy has many of the typical scripting language features in a very Java-like syntax.

To check out the JSR, go to http://jcp.org/jsr/detail/241.jsp. To learn more about the Groovy programming language, visit http://groovy.codehaus.org.

And in Other News

For this month's edition of the JCP column I'd like to draw your attention to two JSRs that recently went final. JSR 127, the JavaServer Faces specification, is final. This specification simplifies the creation and maintenance of user interfaces for server-side applications. It provides a richer graphical composition than is possible with just servlets or JSPs. And to close, the J2ME Web services specification, or JSR 172, has also completed its JCP process. It delivers two optional packages so that nontraditional devices can process XML and make use of and interoperate with Web services.

That's it for this month. I'm very interested in your feedback. Please e-mail me with your comments, questions, and suggestions.

Onno Kluyt is the director of the JCP Program Management Office, Sun Microsystems.

onno@jcp.org

LOOK FOR YOUR FREE...

›Linux ›Java ›Web Services ›.NET ›XML ›Wireless ›Storage ›Security

SYS-CON

The Premier Resource for Today's Corporate & IT Decision Makers

VOL: 1 ISSUE: 1 SPRING 2004

IT solutions ›GUIDE

WWW.SYS-CON.COM/IT

TECHNOLOGIES YOU NEED NOW!

How to Manage Your Ideas Using Today's *i*-Technologies

Top IT Solution Providers!

- › Candle
- › ClearNova
- › Creative Science
- › DralaSoft
- › GraphOn
- › Intersystems
- › iTKO
- › Merant
- › O'Reilly
- › QuadBase
- › SAP
- › Wily Technology
- › Zero G

Coming this ***SPRING!***

MAX: A Java-Based Personal Robot Platform

Paul J. Perrone

When you mention the word "robot," most people think of either large industrial bots that do heavy work on factory floors, suicidal bots doing battle on TV, fanciful R&D bots gracing the labs of universities, or simple hobby bots of the LEGO Mindstorms ilk. Don't get me wrong, all such creations are deeply fascinating to me and embody tremendous ingenuity and engineering craftsmanship. However, I want a robot that I can make do a variety of things around the house or place of business, one that won't cost a small fortune and is flexible and evolvable enough that I'm not stuck with a robot constrained to accomplishing a single task. Essentially, I want something that doesn't exist. I want the "Personal Robot."

Despite what may have been perceived as blind fantasy, a few years ago I set about pursuing the construction of a personal robot. I had a formal background in all of the requisite engineering skills including robotics and AI, but they were festering on my brain's vines of unused knowledge. My years spent doing work that actually generates rather than consumes money thrust me into the world of more commercially viable enterprise technologies and products. However, I began to see more and more of a possibility of fusing the commercial-grade work I was doing with my robotics R&D to yield a robotics platform that was cheap yet sophisticated. I wouldn't need to mortgage my house to buy a platform that only the wealthiest of companies or agencies could afford, and I wouldn't be stuck with a constrained hobby bot that could maybe flicker a few lights and fall down the stairs. Instead, I'd commence building a platform that was based on open source and commercial-grade products, be able to integrate with an assortment of underlying useful mechanical hardware components, and provide myself with an open sandbox and playground in which I could develop a wide variety of robotics apps.

I scrambled to give this platform a name and requisite acronym and could only muster the three letters M-A-X. MAX stands for Mobile Autonomous X-Bot. The mobile and autonomous part should be obvious, but the X-Bot represents the fact that the bot would be a generic platform atop of which I could do cool things. *X* is a variable. Get it? Just plug whatever application you want into the MAX platform and that is what *X* becomes. Armed with a name and an acronym, I was ready to crank up the R&D a notch and began forming a company around said efforts.

There is a plethora of hardware components on the market useable in robots ranging from the capriciously cheap to the stylishly pricey. To create MAX in such a way that would make it easy to work with this varied hardware, we defined a Java-based abstraction layer to interface with such devices. This was a key design feature since as you create or load different robotics applications into MAX, you may also require different robotics hardware to support your application. Our design thus called for plug-and-playability achieved by either custom configuration of a few generic drivers or by creating drivers specific to the hardware device being utilized.

As a Java zealot, everything I aim to do is in Java. If I could Java-enable my nose-hair trimmer, I'd do it. Blind loyalty to programming languages aside, I naturally think there are genuinely good reasons for using Java to build robotics applications. Apart from its simplicity as well as operating system (OS) and computing hardware independence, the wide range of built-in, commercial, and open source tools readily available make it an attractive and low-cost platform for cranking out robotics apps. Thus, we were hell-bent to try and use a standard J2SE runtime operating on Linux (or any other OS) running on a standard CPU for our underlying platform. In fact, this standard JOC (i.e., Java-OS-CPU) platform is indeed the only platform required by MAX for a wide range of robotics applications.

With a standard JOC platform, MAX communicates with external robotics hardware devices via Java-based interfaces to standard parallel, serial, and USB computer ports. For certain devices like digital cameras, the physical connectivity is easy since most cameras plug directly into a USB port. To interface with certain other sensor and actuator types, such as IR sensors and DC motor-driven wheeled bases, a few extremely simple and low-cost hardware connectors are used to bridge between standard computer ports and external robotics hardware.

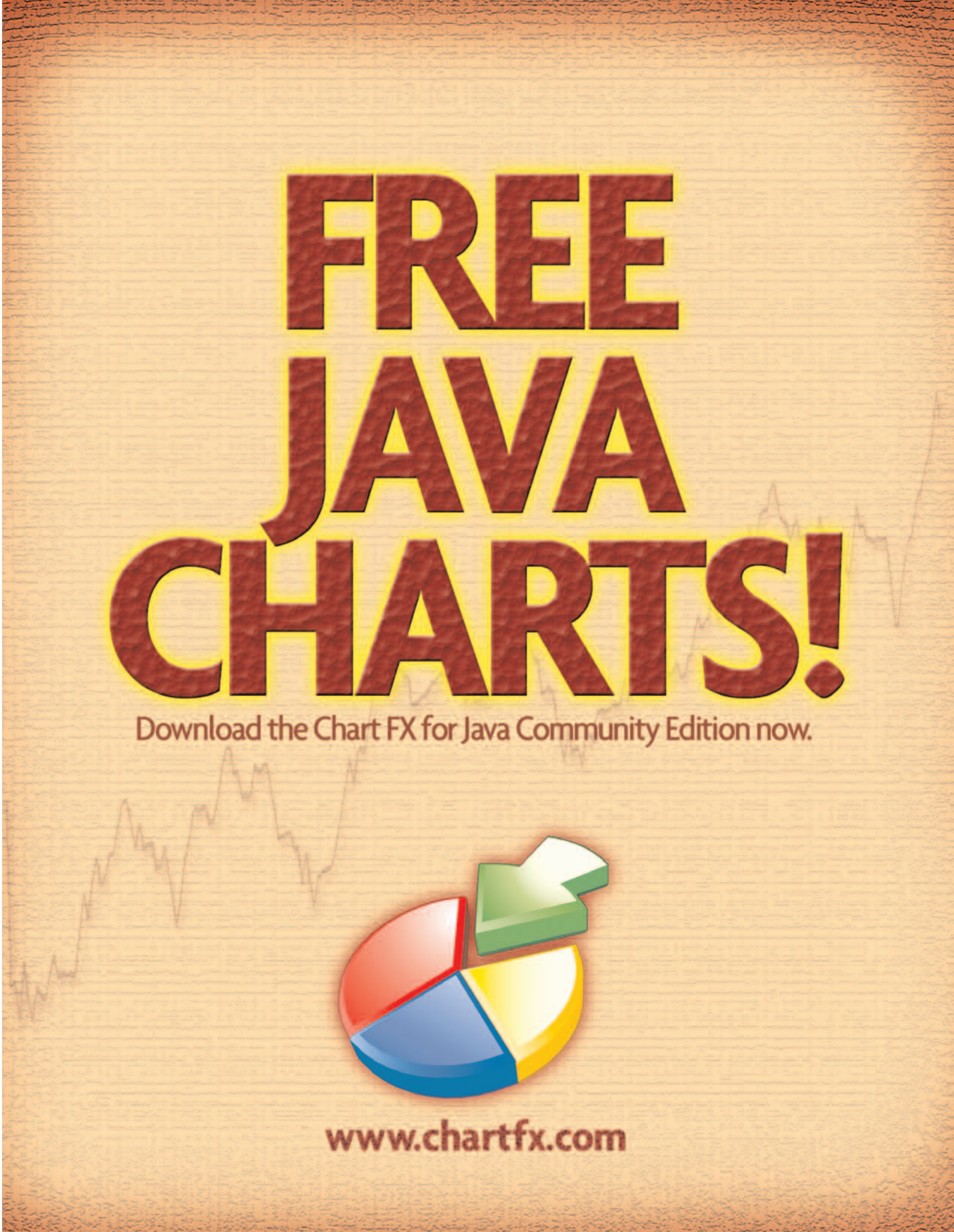
For certain sensors and actuators, a more real-time interfacing approach is required; a standard JOC platform won't cut it here. Hence our MAX design required an augmentation to work with a J2ME-based, real-time Java embedded JOC platform. A few reasonably priced platforms on the market exist with some even built around the aJile Java-based processor core executing Java instructions in hardware.

For a certain range of applications, the embedded JOC platform and MAX profile will be all that is needed to perform a variety of basic robot functions. For other applications, a standard JOC platform and MAX profile will be required for more sophisticated applications that perhaps require access to large amounts of persisted data or communication with multiple distributed services, or need to implement more complex planning algorithms. Yet another alternative is to use a standard JOC MAX profile for the more complex data, communication, and planning operations and leverage use of the embedded JOC MAX profile for the sense-actuate functionality inside the same physical bot.

The MAX software and hardware bridges are designed with maximum flexibility for allowing a choice of underlying hardware while providing open and configurable interfaces for building robotics applications that use and drive such hardware. While a growing suite of MAX components is being built and prototyped with a mixture of hardware devices, the real test will come when we start shipping MAX hobbyist and developer's kits. Only then will we all see what sort of community grows to extend MAX and satisfy the "X equals to" equation, making bots and applications that do things we've always dreamed of and have yet to dream of doing for ourselves and organizations in an open, extremely low-cost, and constraintless fashion.

Paul J. Perrone is an architect, author, and speaker on Java, J2EE, and XML via Assured Technologies, Inc. (www.assuredtech.com). Paul founded Assured Technologies in 1998 and has more recently founded Perrone Robotics, Inc., (www.perronerobotics.com) to focus attention on the vertical space of software for robotics.

pperrone@perronerobotics.com



SPEND LESS TIME PROBLEM SOLVING… AND MORE TIME DEVELOPING APPLICATIONS.