SYS-CON

C#: IS THE PARTY OVER? PAGE 32

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

JDJ

JDJ.SYS-CON.COM VOL.10 ISSUE:8

AOP TECHNOLOGY UPDATE

A framework comparison and internals tour

RETAILERS PLEASE DISPLAY
UNTIL SEPTEMBER 30, 2005

$5.99US $6.99CAN

PLUS...

- Setting Up a Java Shop
- Is Mobile Java a Reality?
- Reflections on Debugging
- Magic Is Golden

Your potential. Our passion.™

Microsoft

JUNK MAIL. MISSING THE TRAIN. ~~APPLICATION INTEGRATION.~~ HAIR LOSS. BAD MUSIC. PAPER CUTS.

WINDOWS SERVER SYSTEM WITH .NET MAKES APPLICATION INTEGRATION LESS PAINFUL.

The Microsoft® .NET Framework, an integral component of Windows Server System,™ is the development and execution environment that allows different components and applications to work together seamlessly. That means applications are easier to build, manage, deploy, and integrate. The .NET Framework uses industry standards such as XML and Web Services which allow enterprise applications to be connected to infrastructure of any kind, which removes many of the headaches of the integration process. Find out more about simplified application integration with Windows Server System and .NET: Simply get the Connected Systems Resource Kit at microsoft.com/connectedsystems

WHEN THINGS GET UNPLEASANT, JUST RELAX AND THINK OF EASIER APPLICATION INTEGRATION.

Microsoft® Windows Server System™

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.

It's Official: Welcome to the 'Technology Bounce Back'

Jeremy Geelan

All the myriad commentators who monitor Internet technologies and the *i*-Technology companies on the NASDAQ doubtless have their own private cluster of indicators that they use to take a weather-check on the overall state of the industry. For some, it's as simple as looking at the NASDAQ index level. This (wholly understandable) approach is the one adopted by SYS-CON's own Roger Strukhoff, who wrote recently:

After going over 5000 at the height of the dot.com bubble, we all know that it plunged precipitously and consistently for the next 18 months. Any hope of a quick recovery was dashed by 9/11, and then a new flicker of hope was extinguished when war came in March 2003. Since then, the NASDAQ's most important numbers have been 2000, 2000, and 2000. The first of the three numbers represents the year of its peak, the second the level at which it settled, and the third the level at which it is apparently going to stay forever.

For others, it's more a matter of looking for the "million tiny points of light" that together will arguably provide more warning of better times ahead than any mere stock index can ever do. And this, you will have guessed, is the tack I prefer to take.

Recently I've become increasingly convinced that, even though at this writing HP has just cut 14,500 jobs and four research projects from its HP Labs organization, those points of light can be found everywhere – not only in the *i*-Technology giants like Microsoft and the middleweights like Macromedia but also, perhaps the best indicator of all, among the renewed trickle of start-ups.

Consider Microsoft (or Sun, for that matter). Redmond has stepped up its pace of hiring, adding nearly 4,400 employees worldwide, and plans to continue expanding at about the same rate in the year ahead. Sun, in the meantime, has – as Ajit Sagar discusses in this issue – started to put its cash hoard to good use by buying SeeBeyond.

Macromedia, poised to consolidate its world-class product set into Adobe this fall, if the stockholders of both companies give the go-ahead next month, has proved that its innovative and customer-focused approach to Web technology results in ever-increasing millions of users being empowered to create truly sophisticated interactive content that was never before possible.

Just as it's a great time to be a developer – as ***JDJ***'s Yakov Fain reports, "*If last September I was calling the Java job market healthy, today's market is hot*" – it is again a great time to be running (or getting hired by!) a start-up. There are several dozen companies now whose upward trajectory has followed that of founder Murugan Pal's SpikeSource, which "productizes" open source software. SpikeSource began only a year ago by closing a $12 million Series A funding round and has not looked back since.

Likewise consider how *LinuxWorld Magazine* editorial advisory board member Andy Astor has, with his co-founder Dennis Lussier, recently launched EnterpriseDB to leverage PostgreSQL's community and BSD license into a PostgreSQL-based database that aims to out-Oracle Oracle (no less). A start-up, however experienced its management team, doesn't just take aim at a multibillion-dollar company like Larry Ellison's powerhouse on a whim; on the contrary, like SpikeSource, EnterpriseDB is convinced that its timing is technologically propitious and that its economic prospects are bright.

On a smaller scale, ***JDJ***'s Jason Bell has in very hands-on fashion demonstrated his faith in the future of *i*-Technology by founding a B2B auction site for the airline industry. It's based on 100% Java and uses as many open source libraries as possible so there is no major financial outlay; even so, what Bell is doing is lighting a candle rather than cursing the darkness. "*Let* the NASDAQ stay flat at 2000," he is in effect saying, "but that doesn't mean the time is not ripe for an agile new technology enterprise to flourish."

When a million Murugan Pals, Andy Astors, Dennis Lussiers, Jason Bells, and Yakov Fains light such candles simultaneously, the glow soon becomes visible to even the gloomiest of prognosticators.

As the old joke goes, "Prediction is very difficult, especially if it's about the future." But some of us – and I am one – are convinced. The technology bounce back has already begun, and is in full flow all around us.

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

jeremy@sys-con.com

ALTOVA®
xmlspy®
2005

Bring your
development
plans to light

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

Revealed in Version 2005:

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

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

Download XMLSpy® 2005 today: www.altova.com

ALTOVA®

Also available in the Altova XML Suite.

AUGUST 2005 VOLUME:10 ISSUE:8

JDJ contents

JDJ (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: JDJ, SYS-CON Publications, Inc., 135 Chestnut Ridge Road, Montvale, NJ 07645.

Ajit Sagar
Contributing Editor

The G.E. of Software

At JavaOne this year, one of the biggest announcements (albeit this one had nothing really to do with Java) was the acquisition of SeeBeyond by Sun Microsystems. It looks like Sun is putting its cash, which it has plenty of, to good use. As we have seen over the last decade of Java, Sun is not really a poster child for making money from software sales. The SeeBeyond acquisition seems to indicate a shift in paradigm, an attempt to drive a stake into another tier (SOA) of the multi-tier enterprise application stack, a way to expand the customer base, and perhaps make some money on software.

The market is developing in an interesting way for vendors who are providing the stack around Java as the platform is being increasingly applied to enterprise solutions. Currently if you look at the market for J2EE (or J EE as the old Java pig with a new lipstick is now called), the application server space has consolidated around a handful of vendors – IBM, BEA, Oracle, Sun, and JBoss. Of these, IBM and BEA are actually the ones that have install bases, which have been used for enterprise applications in large corporations. Oracle is late to the market and is trying to play the catch-up game. JBoss has not yet made a dent in large organizations. And Sun's app server (rebranded for the *n*th time to "Java System Application Server") has really made no dent in the market, in line with all the previous incarnations. Others such as Pramati are looking at partnering with SIs (system integrators) instead of competing directly in the market.

Besides having the best technology, the real play comes down to owning the right pieces of the stack. What is the solution stack on which a company will build portfolios of Java enterprise applications? Let's start from the bottom of the stack. First of all, there is the hardware. The OS runs on the hardware. The database runs on the OS. Software platforms, in this case Java, run on the OS. The application server runs on the software platform. The app server typically integrates through three main mechanisms – synchronous APIs (such as RMI.IIOP), messaging, or HTTP/SOAP (Web services–based integration). Other products, such as a BPM engine, Portal Server, Business Rules Engine, etc., run on the application server foundation. And finally, a Web server makes the application accessible on the Internet.

Of course, this is a simplified view, and there are many more building blocks that lay the foundation for the architecture. But let's go with this picture in mind and look at the top players in the market. BEA had grabbed the majority of the market share since the early days of Tengah and WebLogic by staying ahead of the technology and providing timely optimization while the Java standards were catching up to the market demands. They grabbed the market opportunity, but are currently between a rock and a hard place. The part of the stack that BEA owns is floating above the messaging infrastructure. Basically, they don't have any products that occupy the DB, OS, or hardware tiers. JBoss is another one in this position, but being open source puts them into a slightly different situation. Oracle does own a substantial chunk of the stack by virtue of their obvious presence in the DB tier.

Now let's take a look at IBM. To me, IBM is the G.E. (General Electric) of computing. They own all the pieces of the stack, from the monolithic mainframes that will live on forever to the smallest devices, to the integration technologies (remember the recent acquisition of Ascential), to the professional services you need to deploy and manage large enterprises' IT. They have it all – a true one-stop shop. IBM owns their *clients* – IT and process. Even Microsoft, which is always the target for a monopoly, does not own it all. Scary isn't it? In many ways, with IBM's foothold in open source, they own a large part of Java technology that is in deployment.

So to take the G.E. analogy, with IBM's offerings, you could pretty much build/buy everything from cars to refrigerators to razor blades from big blue. Eventually others will have to partner, merge, and/or reincarnate to compete successfully. I always wonder where BEA will go next. To Oracle or to Sun?

Ajit Sagar is a principal architect with Infosys Technologies, Ltd., a global consulting and IT services company. He has been working with Java since 1997, and has more than 15 years' experience in the IT industry. During this tenure, he has been a programmer, lead architect, director of engineering, and product manager for companies from 15 to 25,000 people in size. Ajit has served as *JDJ*'s J2EE editor, was the founding editor of *XML-Journal*, and has been a frequent speaker at SYS-CON's Web Services Edge series of conferences. He has published more than 75 articles.

ajitsagar@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 *robyn@sys-con.com*
National Sales and Marketing Manager:
Dennis Leavey *dennis@sys-con.com*
Advertising Sales Manager:
Megan Mussa *megan@sys-con.com*
Associate Sales Manager:
Dorothy Gil *dorothy@sys-con.com*

Editorial
Executive Editor:
Nancy Valentine *nancy@sys-con.com*
Associate Editor:
Seta Papazian *seta@sys-con.com*

Production
Production Consultant:
Jim Morgan *jim@sys-con.com*
Lead Designer:
Tami Lima *tami@sys-con.com*
Art Director:
Alex Botero *alex@sys-con.com*
Associate Art Directors:
Abraham Addo *abraham@sys-con.com*
Louis F. Cuffari *louis@sys-con.com*
Assistant Art Director:
Andrea Boden *andrea@sys-con.com*
Video Production :
Frank Moricco *frank@sys-con.com*

Web Services
Information Systems Consultant:
Robert Diamond *robert@sys-con.com*
Web Designers:
Stephen Kilmurray *stephen@sys-con.com*
Vincent Santaiti *vincent@sys-con.com*

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

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

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

The Developer Paradox:

No time to test your code? But long hours reworking it & resolving errors?

Check out Parasoft Jtest® 7.0
Automates Java testing and code analysis.
Lets you get your time back and deliver quality code with less effort.

- **Automated:**
  Automatically analyzes your code, applying over 500+ industry standard Java coding best practices that identify code constructs affecting performance, reliability and security.
  Automatically generates and executes JUnit test cases, exposing unexpected exceptions, boundary condition errors and memory leaks and evaluating your code's behavior.
  Groundbreaking test case "sniffer" automatically generates functional unit test cases by monitoring a running application and creating a full suite of test cases that serve as a "functional snapshot" against which new code changes can be tested.
- **Extendable:**
  Industry standard JUnit test case output make test cases portable, extendable and reusable.
  Graphical test case and object editors allow test cases to be easily extended to increase coverage or create custom test cases for verification of specific functionality.
- **Integrated:**
  Integrates seamlessly into development IDE's to support "test as you go" development, and ties into source control and build processes for full team development support.

To learn more about Parasoft Jtest or try it out, go to www.parasoft.com/JDJmagazine

PARASOFT®
We make software work.™

Automated Software Error Prevention™

Using Metrics to Optimize J2EE Application Performance in Production

The payoff is substantial

by Brad Micklea
and Geoff Vona

Despite the increasingly widespread adoption of J2EE for enterprise applications, measuring their performance in production continues to be a black art. Without knowing what to look for, many people measure anything that seems useful, which soon results in an overloaded system and reams of meaningless metrics data. It's tempting to just throw up your hands and start making system changes based mainly on hunches.

This article provides a strategy for measuring performance in production. Starting with an explanation of some basic metrics that express the performance of a J2EE application, we'll apply these to a model of a J2EE system, showing how these metrics interact in different areas of the system. From here we will drill-down to discuss the nuts and bolts of gathering metrics from each type of J2EE component, from the client Web browser through to the back-end database. Finally, we'll uncover the quality and cost of different metrics and how to handle that tradeoff so that minimal overhead is imposed on the system being measured.

Brad Micklea is a product manager at Quest Software. Brad has helped companies improve the performance of their Java and J2EE applications for over three years.

Geoff Vona is a development manager at Quest Software. A Java programmer since early 1996, Geoff's current work focuses on J2EE performance monitoring and diagnostics.

Basic Metrics

Some of the most basic metrics for a J2EE system (or any system really) are:

- Response Time (R)
- Throughput (X)
- Resource Utilization (U)
- Service Demand (D)

Response Time is the best overall indication of your end-users' experience. It has a much higher variation than the other metrics so it is essential to understand the distribution of response time. If most users experience two-second response times but 10% are getting 10 second response times, for example, you need to know this in order to assess and fix the problem.

Throughput, the number of transactions executed by the system over a period of time, is a good indication of the system's ability to handle load.

Resource Utilization, or how heavily a particular system element is being used, is the easiest metric to understand. It's not necessarily the most useful for determining system performance because only the utilization of *contended* resources has a significant impact on performance. We have found it more useful to define a new metric called Service Demand, calculated by the following formula:

```
 Service Demand = Utilization of the
resource / Throughput
```

Service Demand looks at resources in terms of the demands being put on them. This gives us a clear idea of the utilization of a resource as the users' demand for it increases.

Interrelating Metrics

These metrics are related, as shown by Figure 1. Understanding this relationship is central to building a model of J2EE performance.

The first thing to notice is that throughput and response time are often at odds. For an interactive application in production, we typically want to maximize throughput, as long as response time is at or below some threshold. For example, we may find we can achieve a maximum throughput of 100 transactions per second while keeping response time at or below two seconds.

Figure 1 also shows that resource utilization typically controls system behavior. It is resource contention that causes the dramatic drop in throughput and the commensurate rise in response time that marks the Buckle Zone. The effect of the Buckle Zone is a severe drop in application performance, due to the system spending most of its time managing resource contention, rather than servicing requests.

It's important to see how your application behaves at each of these three zones and the specific metric values that cause the system to shift from light to high load to buckle zone. This will be particularly useful for setting alerts in your production monitoring tool.

A Model for J2EE Systems

Let's fit these metrics into a model of a J2EE system that has four main processes, illustrated by Figure 2. *Client Handling* includes the originating requests and the sessions those requests may be tied to. *Execution Management* is where

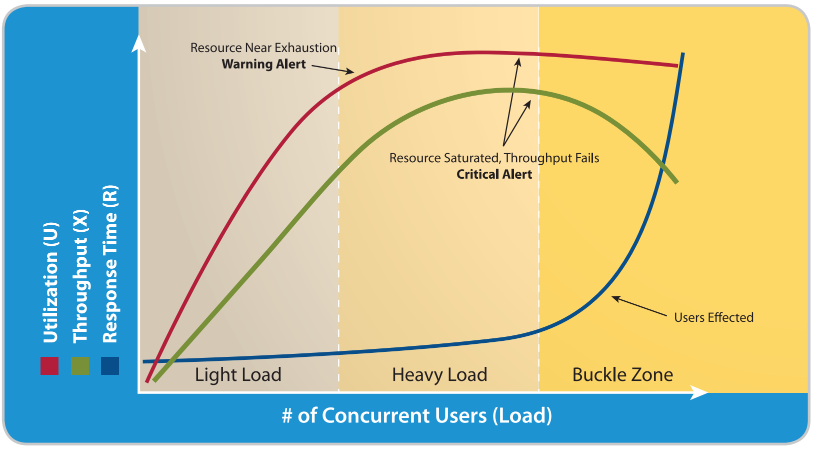


Figure 1 The effects of load on resource utilization, throughput and response time

requests are queued before being assigned to execution threads. *Applications* includes both your code and all the standard J2EE components. *Services* include anything that connects your application to the outside, such as JDBC and JMS. Of course, all these elements run in a *Java Virtual Machine (JVM)* which, in turn, is using the resources of the operating system such as its processor, memory, disks, and network connectivity.

Figure 2 also shows how the metrics discussed earlier can be measured at each level. We'll need to measure and understand response time dispersion at the client-handling level, the application code level, as well as for services like JDBC. Throughput is most important at the client handling level – we have to understand how much user load our system can handle. Resource utilization is measured at many points in the system (OS, execute threads, services, etc) so that we can correlate the information and see how different elements in the system are affecting each other.

Points of Measurement and Overhead

There are no free metrics. Measuring something in real time always adds some overhead; the key is to make intelligent choices about what information to collect while keeping the overhead we're willing to incur in mind. The best way to determine the overhead of a measurement is to use the Service Demand calculation defined earlier. It is through that lens that we will now look at how we can most effectively and efficiently gather measurements from the J2EE system. Figure 3 illustrates the many possible measurement points in a J2EE system.

There are two methods for measuring client response time: browser scripting and injection of synthetic transactions. Browser scripting is usually implemented through JavaScript in the HTML pages returned to users. While the best measure of the user's experience, it does present several significant difficulties: it's very hard to measure all the clients reliably; deployment and maintenance of the scripting code for those pages can become difficult and tedious.

Synthetic transactions address most of these shortcomings and have become more commonly used. The idea is to inject synthetic or scripted user transactions into the system with some tool. These transactions can be easily measured and give a good approximation of what the users are experiencing. It's important to realize their limitations – unless the injector is placed near the end users, rather than just outside or inside the firewall, it cannot provide data on the wider network effects; also, creating realistic synthetic transactions does require a fairly detailed knowledge of user/site interaction and the patience to accurately model this interaction. However, the control that synthetic transactions provide overshadows these limitations.

Operating system (OS) metrics, familiar to most developers, will be gathered from machines throughout the J2EE system. Seeing the shifting patterns of CPU, memory, and disk usage on the various tiers of the system greatly aids understanding. But to build that data into a useful J2EE system model, you have to be able to accurately associate system metric information with application container and application code data at specific intervals in time. Only by doing this can you draw a picture of the complex interactions between the application, its application server container, and the underlying system.

Metrics from other points, such as Web servers and databases, can be treated in the same way as OS metrics. Unfortunately, there's no standard for them; each vendor provides a metrics interface that it feels is appropriate for its offering.

J2EE and JVM Measurement

Operating system metrics should also be correlated with Java Virtual Machine (JVM) performance. JVMs provide some information (largely heap usage) at almost no cost to most application servers, generally through JMX. For more detailed information, turn to the Java Virtual Machine Profiling Interface (JVMPI), which exposes things like object allocations/deallocations, thread locks and call stacks, method and line execution times. However, JVMPI's main drawback is that it is a very intensive interface that imposes a lot of overhead in the system; we cannot recommend its use in production systems. The jury is still out on whether the new Java Virtual Machine Tool Interface (JVMTI) will be appropriate for production use.

Application server metrics are generally quite detailed within the major vendors offerings. Though each server is different, they all include information on the following:

- Response time and utilization for servlets, JSPs, and EJBs
- Caching for EJBs, JDBC connections, and other service elements
- Utilization of services like JDBC, JMS, JCA, and JNDI
- Transactional information (# of rollbacks and commits)
- Threading/queueing data (# of active threads, # of waiting requests)
- JVM metrics (heap)
- General configuration information

These metrics are important because they can provide the response time, throughput, and utilization information you need to complete your J2EE performance model. Also, your ability to gather this data for either the complete system, or portions of it, means you gain a lot of flexibility in your analysis. Of course as is often the case with powerful and flexible systems, navigating the expanse of data can be cumbersome, especially since it is not by default tied to specific application transactions or requests.

BEA WebLogic exposes its metrics through JMX. This standards-based approach can make it much easier for you to gain access to this data through an interface other than the WebLogic console. WebLogic's metrics themselves are quite strong; however, there is a notable lack of information on Response Time for EJB methods (though this information is provided for servlets).

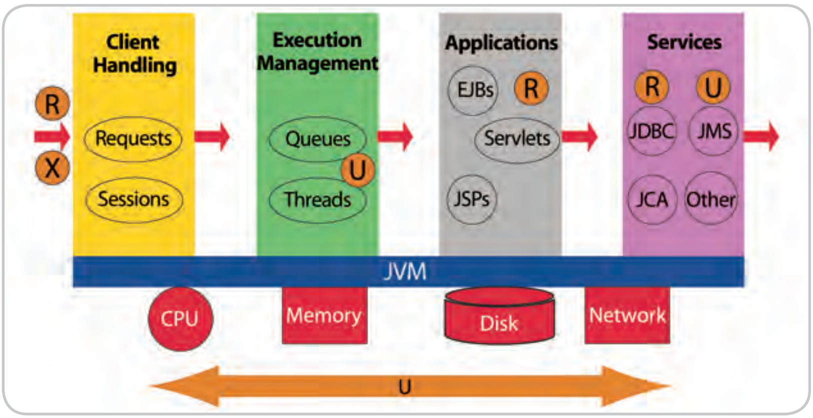


Figure 2 Model of J2EE system with metrics applied

IBM WebSphere in version 4.x used their proprietary PMI interface, which provided slightly more complete metrics than WebLogic, but at the expense of J2EE compliance (i.e., it is not JMX-based). For version 5.x of WebSphere, the metrics are all available either through PMI or through JMX. One other nice feature of the WebSphere interface is the ability to control the detail received. PMI can be set to report back data at none, low, medium, high, or maximum level (note that maximum level turns on JVMPI).

Oracle's application server provides metrics through its DMS servlet. Its data includes information about the performance of JVMs, JServ, JDBC, and EJBs, but is not as complete as WebLogic or WebSphere. The JBoss application server makes its performance metrics available through JMX. Its metric set is relatively complete, and includes data on both Tomcat and the application server.

Fortunately, gathering this data from the application servers does not generally cause a lot of overhead. Though you can't turn off the MBean data collection in WebLogic that provides the JMX metrics, the overhead is usually not an issue unless you're doing a lot of remote MBean queries. WebSphere's PMI interface allows you to turn off data collection, which obviously reduces overhead. However, our tests haven't shown a great deal of difference between the middle three detail levels.

Protocol sniffers can be another source of valuable data. These work by reverse-engineering the protocol that passes between clients and servers. This nearly touchless interaction requires almost no overhead to perform. The data that is provided can be extremely valuable, but it's almost always very detailed and deep in nature, demanding a great deal of expertise on the part of the user to interpret. Also, these solutions do tend to be quite expensive due to the large amount of valuable data they provide with little overhead. If you can afford them and know how to effectively use the information they provide, they are often a good investment.

The final data collection strategy is application code instrumentation, of which there are two types: custom and automatic. Custom instrumentation is handled entirely by the developer by inserting measurement code into the source code or generated bytecode to gather the metrics they want (typically J2EE component- or method-level timing information). This information is then sent on to a centralized recorder that the developer would also have to write. This a very complex undertaking and one that most companies prefer to avoid through the purchase of a transactional J2EE diagnostic tool.

There are several ways to do automatic instrumentation, but regardless of how it's done it saves time by automating the addition of performance measuring code to the generated bytecode of the application classes. Vendors of performance tuning tools have spent considerable time and effort optimizing the overhead of their automatic instrumentation, so it really does offer the lowest possible overhead, even over a custom solution. Automatic instrumentation can also be applied to any application quickly and easily, while custom instrumentation requires considerable time and effort on the part of the developers to add the necessary code.

Regardless of how this data is gathered, it provides an excellent view into your application code – something that none of the other metrics techniques can do. Typically you can do some or all of the following metrics on a J2EE component-, class- or method-level granularity:

- Call counts
- Time spent in a method (Exclusive Time)
- Time spent in a method and all the methods it calls (Cumulative Time)
- Exceptions thrown
- Bytes transferred / serialized in RMI
- Stack information

This deep application information is essential for quickly and correctly diagnosing many code-related performance problems. It can help you to determine where the application is being misused or is breaking down, as well as easily isolate the method-level bottleneck that may be choking your application under production load.

Applying It to Your J2EE Application

Unfortunately, there is no single model that will work with all J2EE systems – each is, to some extent, custom and must be dealt with individually. Investigating performance issues in production means working through a systematic analysis for each case.

Begin by ensuring that the application stakeholders are clear on what level of performance they require. Then decide what information you will need to evaluate the current system performance. This is where you need to strike a balance between the amount of information you want and the amount of overhead you can incur. From that point you can move on to deciding how to gather the information you need and what kind of a performance model your system will adhere to.

All of this does require effort – you'll have to be diligent about controlling system variables by locking down the environment and re-baselining to ensure that you're not measuring on a shifting foundation. But the payoff from this effort is substantial. With these guidelines and common sense, you will be able to release an application that exceeds its performance expectations and is more reliable as well. All of this spells greater application confidence, which is something any business would love to have more of.

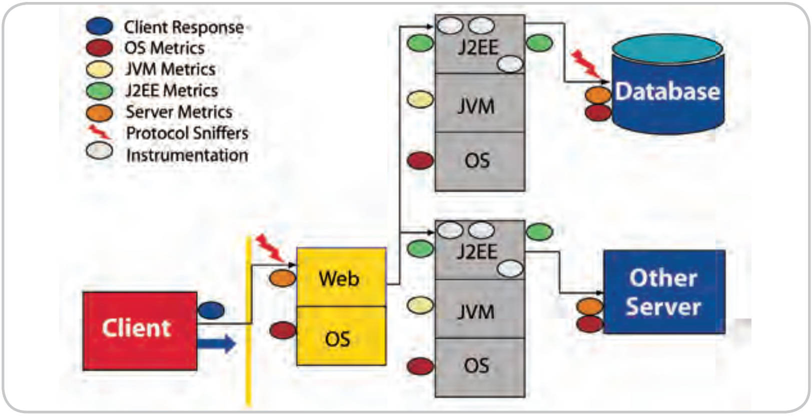


Figure 3 Measurement points in a J2EE system

Perforce SCM.
Bring your local heroes together.

Perforce Software Configuration Management brings you and your team together - wherever you happen to work.

[Fast]

[Scalable]

[Distributed]

You've got developers in New York, London and Singapore collaborating on your project. Your UI designer is in San Francisco and your QA team is in Buenos Aires. SCM nightmare? No. Just another day's work for Perforce.

Perforce Proxy, part of Perforce's multi-site solution, brings files to remote users on demand and caches them locally for other users at the same location. Local caching guarantees quick response times for remote users while maintaining real-time access to project activity and status information.

Unlike replication solutions with their inherent administrative overhead, a Perforce multi-site solution recovers files automatically, reduces the load on the Perforce Server, and requires no remote backups.

Fully integrated into Perforce's client/server architecture, Perforce Proxy requires only TCP/IP connectivity. A single Perforce Server can support any number of Perforce Proxy sites, allowing developers all over the world to collaborate.

PERFORCE
SOFTWARE

Download a free copy of Perforce, no questions asked, from www.perforce.com. Free technical support is available throughout your evaluation.

AOP Technology Update

A framework comparison and internals tour

by Patrick Fendt

Aspect-Oriented Programming (AOP) is undeniably one of the coolest things to happen in the software technology in a long time. AOP has been called the "third dimension of programming" (copyright by Frank Sauer, Technical Resource Connection, Inc.) and has tremendous power in dynamically inserting logic into pre-existing programs. It can help solve some of the key problems (technology gaps, *so to speak) still facing IT organizations. More specifically, AOP is now beginning to* bridge the gap *in three areas of software technology:*

1. Separation of concerns – so developers can focus on the business logic they need to design, implement, and test.
2. Making IDE-type tooling simpler and more standards-based (e.g., for Model-Driven Architectures, code-generation, deployment artifacts-generation, etc.).
3. Runtime management of software systems – including profiling, auditing, and trouble-shooting.

The purpose of this article is to give the reader an understanding of the various AOP technologies available today and put AOP in perspective so it's judiciously used. But first let me give you a brief overview of AOP and AOP terminology. The source code for this article can be downloaded from www.jdj.sys-con.com.

Patrick Fendt is a presales technical consultant at BEA Systems. He spent the first seven years of his career designing and developing high-performance middleware at IBM, Scientific Software, and S2 Systems. He has been in the technical/pre-sales organization at BEA for over six years – serving in various roles. Patrick graduated with honors from Georgia Institute of Technology with a BS in computer science

pfendt@bea.com

AOP Overview and Terminology

OOP took the first step toward separating concerns in letting software designers compartmentalize their code – both logically and physically. OOP also supports polymorphism so designers can generalize their code in such a way that functionality can be applied (and hence reused) to various subtypes without knowing in advance which specific sub-type is being used. This aids developers in designing more extensible and manageable software, making their code more reusable.

AOP takes this concept a step further and lets this *separation of concerns* be applied easily to a class of functions that tend to permeate a software system; examples are logging, security, transactions, exception handling, and monitoring. These are examples of what we call *crosscutting concerns*, and they tend to be extremely critical to the success of a software system – especially over its lifetime. Yet, until AOP came on the scene, these crosscutting concerns caused enormous inefficiencies, inconsistencies, and bugs, and as a result, these systems typically fell short of expectations on quality, manageability, and extensibility.

AOP technology introduces several new terms to our software technology vocabulary. In essence, it allows completely independent Java methods (*advice*) to intercede transparently and be executed while running pre-existing Java application code – at well-defined points called *join points*. Examples of join points are before and after method calls, or when a particular object is instantiated, or when a particular member variable is referenced. The way developers specify which join points are relevant is via a *pointcut* expression that refers to one or more join points in a particular software system or subsystem. A pointcut could, for example, apply to a collection of Java classes in a directory hierarchy, a JAR, or an EAR, and the expression typically takes advantage of your class naming conventions and uses a regular-expression-like syntax. Finally, an *aspect* is a more general term referring to one or more related advice functions. An aspect can then also serve as a focal point for specifying the pointcut(s) that relate to the aspect. What follows are some commonly used AOP terms:

Join point:	a well-defined point in programs where AOP functions (advice) can intercede and affect execution
Advice:	the action taken by the AOP framework at a particular join point. This functionality (implemented via a Java method) will transparently intercede and be executed while executing your Java application code. Examples of different types of advice are: *before*, *after*, and *around*
Aspect:	a general term referring to a collection of one or more related advice methods, typically implemented in a single Java class, and class, and potentially accepting join point context information as a parameter to those advice methods

Pointcut:	the expression that specifies which join points will be affected by one or more aspects
Introductions:	a more intrusive type of intercession that modifies the structure/type of your (Java application) classes, for example, by implementing a new interface or adding a new member variable
Mixin:	a *mixin* refers to adding new class(es) to a pre-existing class
Weaving:	the application/activation of an advice across one or more pointcuts so that the advice will be executed when the join points are encountered
Target:	the application object being advised (affected by the aspects) at any point in time
Interceptor:	an aspect with only one advice method named "invoke"

AOP Use-Cases

Now let's briefly consider some typical use-cases for AOP. Candidate functionality would be any horizontal/crosscutting type functionality (required across multiple applications and multiple subsystems) that architects and/or application developers are often faced with implementing. Some common examples are:
1. Logging, tracing, auditing
2. Security authentication and authorization
3. Rules engine
4. Testing and test-harness support
5. Persistence and O/R mapping
6. Caching: caches objects transparently to the application developer
7. Transactions support
8. Synchronization and thread safety
9. Exception handling

As you can imagine, implementing these kinds of system-wide features once, perhaps in a single class or subsystem, would have tremendous ROI and cost-savings benefits for IT organizations. It should be noted here that the application infrastructure (middleware) addresses many of the crosscutting requirements listed above; however, some of them are only partially addressed by middleware (e.g., one example is testing), and standards-based middleware is currently largely API-driven.

AOP Technologies and Frameworks Comparison

AOP frameworks can be classified according to the technologies they employ to implement AOP. Fundamentally, AOP implementations differ based on how they approach and implement the following:
1. Join points and pointcuts
2. Aspects
3. Weaving (build-time and/or runtime)
4. Tooling: level of IDE integration and/or monitoring

What follows is a description of these technologies and a mapping of which ones the major AOP frameworks use. The AOP frameworks we'll consider are: AspectJ, AspectWerkz, Spring, JBoss AOP, and a homegrown approach (that we'll call *DARAC* short for Dynamic AOP using Reflection, Annotations, and Controls) that I developed to illustrate the advantages of implementing AOP using annotations, reflection, JMX, and Java controls.

Note that *Controls* refers to an Open Source technology for reusing Java that was introduced by BEA Systems as part of the Apache Beehive project. For more information, see the WLDJ article from November 2004 entitled "Open Source Technologies" at http://wldj.sys-con.com/read/47092.htm.

Finally, I should point out that DARAC is presented here simply for illustrative purposes. DARAC wasn't designed to be a production-ready implementation.

Now we'll compare and contrast the major AOP frameworks, and describe the functionality and internals of the DARAC approach for two reasons:
1. DARAC will help us understand one perspective on how AOP can be made dynamic and how AOP relates to other Java technologies
2. It's informative and beneficial for architects to understand the internal mechanisms of AOP so that they can make a more intelligent decision on which AOP framework best suits their needs.

AspectJ (version 1.2)

AspectJ is a code-based and pre-compiling-based approach to AOP. In other words, the aspects and pointcuts are expressed in a non-Java source file, and then a pre-compiler is run against the code to generate the standard Java artifacts required for a typical ANT-based build. For developers who are willing to take this approach, it obviates the need for a separate XML file to express pointcuts, and it is a more centralized approach where the aspects and pointcuts can be combined in a single file. However, it comes at the cost of introducing language extensions and intruding on your build process. To configure which aspects apply to the subsystem you're working on, you specify a list (a *.lst* file) to the compiler or IDE. As a result, with respect to weaving, AspectJ is oriented toward build-time weaving (there's no inherent support for turning aspects on/off at runtime). Finally, IDE support in AspectJ is fairly advanced – including AOP support via an AJDT Eclipse plug-in. This plug-in supports the following features:
1. The AspectJ compiler automatically detects syntax, grammar, or spelling errors in your aspect code during development/editing
2. It calls out what aspects and advice are in effect
3. It outline window shows what application code is affected by all the advices in effect
4. All affected join points are highlighted to show that an advice is in effect for them

This first IDE-integration feature (development-time syntax and grammar checking) is perhaps the strongest argument for using AspectJ as opposed to other AOP frameworks.

AspectWerkz (version 2)

AspectWerkz takes a significantly different approach to AOP than AspectJ. It doesn't rely on a pre-compiler, but instead makes use of either annotations or an XML file (*aop.xml*) to specify pointcuts. The underlying intercession technology is JVM-based bytecode manipulation – either at build-time or load-time (though they also support an older proxy-based approach similar to Spring AOP).

Since they use JVM-based intercession, the aspects themselves can be normal Java classes, and the advice is implemented as methods on those aspect classes. AspectWerkz also leverages the *aop.xml* file to specify what aspects are being applied to a particular system/subsystem. With regard to weaving, AspectWerkz supports both build-time and runtime. Note that when I refer to *runtime weaving*, I'm implying the ability to turn AOP on/off at runtime (preferably on a per-pointcut and/or per-aspect basis). Again, AspectWerkz achieves this via JVM-level functionality (e.g., JVMPI or JVMTI and a "hotswap" architecture). For IDE support, AspectWerkz provides some basic NetBeans 3.6 support (to support weaving); it also provides an Eclipse plug-in supporting the following features:

1. Javadoc-style annotation support for Java 1.4-based AOP using AspectWerkz in annotations mode
2. Logging/tracing of AspectWerkz-specific information
3. Highlighting join points (in the editor) with the ability to jump to the applicable advice code

In summary, AspectWerkz is a very powerful AOP framework. One of the only drawbacks would be the lack of development-time error checking with respect to pointcut declarations, and the only framework that fully supports this would be AspectJ. The good news is that AspectJ and AspectWerkz are merging their technologies. One of the first deliverables is an offshoot of AspectJ that provides annotations-based AOP under the project name *@AspectJ*.

Spring AOP (version 1.2)

Spring AOP provides a dynamic proxy-based approach to implementing AOP (using Java reflection APIs). While this approach makes introductions more difficult to implement, it has one important advantage – proxy-based AOP is 100% pure Java-based. In other words, there are no dependencies on either a pre-compiler or any JVM-specific features.

One disadvantage to this approach is that the target application classes must implement some sort of interface to be supported by this dynamic proxy-based approach. However, this restriction is easily overcome by using the CGLIB Open Source toolkit (CGLIB is used to extend Java classes and implement interfaces at runtime) to extend the Java Proxy mechanism to application classes without interfaces. Moreover, doing this is simple and well documented.

With regards to specifying aspects and pointcuts, Spring AOP relies more heavily on XML – using the *springconfig.xml* file for both Spring Framework and Spring AOP configuration information. Advice is specified via Java code in one or more of the following types of classes:

- ***MethodInterceptor:*** implements *around* advice – adheres to AOP Alliance signature for method interception (see http://sourceforge.net/projects/aopalliance for details)
- ***Advice:*** any advice class implementing one of the basic Spring AOP advice interfaces (before, after-returning, and throws)
- ***Advisor:*** encapsulates an aspect with both a Java class and an associated XML-based definition by specifying both a pointcut and the associated advice

Note that both pointcuts and advice are implemented in concert with Spring's IoC (Inversion of Control) framework. This is good news for those already familiar with the framework, and Spring AOP is essentially implemented as an extension to this IoC core functionality. Spring AOP is also extensible in that custom pointcut classes and custom advice types are supported. Moreover, because of its integration with the Spring framework, Spring AOP comes with some level of built-in transaction and security support, and other types of aspects would also be easier to implement if support already exists in the base framework. With respect to weaving, Spring supports the runtime approach. Finally, it should be noted that BEA recently announced a level of commercial support for the Spring framework.

JBoss AOP (version 1.1)

JBoss is fairly similar in general to the AspectWerkz framework. It supports both annotations-based and XML-based approaches to specifying pointcuts, advice, and aspects (including a bind primitive to bind a pointcut expression to an interceptor method or advice/aspect class implementation). Moreover, both AspectWerkz and JBoss AOP support explicit binding of pointcuts to advice.

In JBoss, with the XML-based approach, pointcuts, advice, and aspect mappings are specified in the JBoss AOP XML file (*jboss-aop.xml*). With the annotations-based approach, you can use either Java 1.4 or Java 1.5. A JBoss aspect can be any Java class (but it must have an empty constructor). JBoss AOP also supports a *scope* primitive associated with aspects and interceptors. The scope specifies how many instances of the aspect class will be instantiated per-JVM, per-class, or per-instance.

And like the other frameworks, JBoss AOP supports dependency injections, introductions, and mixins. With respect to weaving, JBoss AOP supports hot deployment of aspects (and runtime weaving). This means that advice bindings can be added or removed at runtime for the target classes that were included by one of the following primitives: *pointcut*, *bind*, and/or *prepare*. Moreover, the JBoss management console can be used to effect the runtime weaving.

JBoss AOP provides an Eclipse plug-in with the following features:

- Define interceptors
- Right-click on target methods to apply interceptors to targets
- Auto-generates jboss-aop.xml file
- Support for markers indicating a particular target's method(s) and/or field(s) are advised
- Advised members view to show all the advised members for a specific target class

"AOP is now beginning *to bridge the gap* in three areas of software technology"

Formula One e.Spreadsheet Engine

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

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

Excel-enable your Java applications

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

Embed live, Excel-compatible data grids

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

Automate complex calculations and rules

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

Read and write server-based Excel spreadsheets

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

Make spreadsheets a part of your strategies

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

ACTUATE ReportingEngines™

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

FREE TRIALS, DEMOS AND SAMPLE CODE

- Aspect Manager window to view all the bindings and pointcuts in effect (support changes)

Finally, it should be noted that like Spring AOP, JBoss AOP provides several out-of-the-box aspects. Examples are aspects for implementing thread-local member variables and caching.

DARAC

The DARAC framework gives us an alternate view of AOP and facilitates a discussion of AOP internals. One of the most significant differences between the previously discussed frameworks and DARAC is that the latter uses Java control(s) to specify aspects, and each method on the aspect control specifies the associated advice type, aspect class, and pointcut expression. Multiple methods can then be added to each control so that multiple combinations of aspects, advice, and pointcuts can be specified for each control.

DARAC employs Java reflection and the dynamic proxy as the intercession mechanism in the Java application. As a result, DARAC uses a factory pattern to instantiate application objects that will participate in the AOP framework (refer to Listing 1 for an example of creating a proxied target object).

As with the Spring AOP framework, the CGLIB could be used here – so that these application classes wouldn't have to use interfaces. Aspects classes aren't restricted and any Java class can serve as an aspect. Weaving is controlled via a JMX listener, and so aspects can be turned on and off dynamically at runtime using a JMX-based administrative command. DARAC was developed for WebLogic Workshop 8.1.

DARAC Internals

To better understand AOP technology let's take a look under the hood so to speak. Listing 1 shows application code instantiating an AOP-enabled POJO object. DARAC makes use of a Java control (implemented with WebLogic Workshop 8.1 using Java 1.4 and javadoc-style metadata annotations) as the primary mechanism for introducing AOP into the application and the associated IDE project. Note that in the next major release BEA intends to deliver the Workshop IDE as a suite of Eclipse plug-ins rather than a separate IDE. This AOP control is implemented via the *AspectManagerImpl.jcs* class, which can be downloaded from http://jdj.sys-con.com. This class serves the following purposes:

1. If necessary, it instantiate the global (per-JVM) *AOPManager* singleton class
2. Adds the aspect definitions into memory via *AOPManager*
3. Serves as a factory for applications to instantiate AOP-enabled objects; as shown in Listing 1, the *createProxy()* method is used to instantiate these objects

This control supports method-level metadata-based attributes for specifying the aspect information. The method signature serves as a placeholder and specifies the name of the advice to invoke in the aspect class (if no such method exists, then the *invoke* method is called). The supported attributes are:

- ***Aspect:*** the name of the aspect class
- ***Advice:*** specifies the type of advice (before, after, around)
- ***Pointcut:*** the regular expression used to match against a fully qualified target class name
- ***Enabled:*** the boolean indicating whether the aspect/advice is enabled (by default) at startup-time or not

The *createProxy()* method contains the code that associates the Java dynamic proxy with the application interface/class. This code uses the *AOControlIH* class as a lightweight invocation handler to be passed to *Proxy.newProxyInstance()*.

This *newProxyInstance* method is how Java reflection supports method-based intercession via a callback into the invocation handler (a.k.a., the *dynamic proxy*). Whenever the target object's methods are called, Java will first invoke the *invoke()* method on the invocation handler class. As a result, the Java dynamic proxy in this case is the AOControlIH class. However, my implementation of the invocation handler is simple; it just forwards the *invoke()* call to the global AOPManager class. This AOPManager class contains the intelligence as far as what aspects exist, what targets are active, and whether the aspects are active at any point in time. AOPManager maintains three critical data structures:

- HashMap of aspects – keyed by aspect class name
- HashMap of targets – keyed by target class name
- Targets[] array for efficiently maintaining a correlation between target objects and their respective/advising aspects

The AOPManager is best explained by taking a look at its key methods. The *getAOPManager* method serves as both an access mechanism to retrieve a reference to this singleton object and as a factory to create the singleton. Also note that during initialization of the singleton the AOPManager class is bound into the J2EE JNDI tree so that:

1. It registers its existence to the administrative console
2. It avoids garbage collection (being a per-JVM global singleton)

The *addAspect* method simply adds aspect information to a new HashMap entry for this aspect, and then adds an associated MBean object representing the state of this aspect. Not surprisingly, the *addTarget* method adds the target object to the targets HashMap and initializes a *targets[]* array entry for this new target. The *addAspectsToTarget* method then correlates these two data structures by iterating across the aspects HashMap and for each aspect it checks to see if the new target matches the pointcut expression for that aspect. If so, then the *addAspect* method is called for the AOTarget associated object.

The meat of this AOP implementation is in the *invoke* method in this AOPManager class. This is where the actual intercession occurs: the AOControlIH class proxies its *invoke* method call to this *AOPManager.invoke()* method, which does the following:

1. Locates the AOTarget object associated with this interceded/target object
2. Loops through the applicable aspects
3. If the aspect is enabled, it queries the advice type, and if appropriate, invokes associated advice method before and/or after invoking the target method

Deploy *Intelli*VIEW... and you can be the one!

IntelliVIEW is an easy to use interactive reporting solution that allows IT to create, deploy and distribute reports in a matter of minutes.

With IntelliVIEW you can:

- Connect to any database
- Generate ad-hoc reports
- Provide powerful analytics
- Integrate seamlessly into web applications
- Define user access levels
- Schedule and email reports automatically

These combined with its uncomplicated licensing and low TCO give you the power to be the IT Super Hero of your organization.

Be the IT Super Hero of your organization

call toll-free **1-866-99IVIEW**

or visit

http://www.intelliview.com/jdj

Product of Synaptris

3031 Tisch Way, Suite 1002, San Jose, CA 95128. USA

Framework	Join Point & Pointcut Approach	Aspects Implementation	Weaving Approach	IDE Tooling (Eclipse plug-ins)	Interception Technology
AspectJ	Via code	Highly specialized classes using AOP-specific Java extensions	Build-time orientation	Excellent	Uses pre-compiler
Spring	XML file	Implement AOP-specific interfaces	Runtime orientation	Minimal	Uses reflection (dynamic proxy)
AspectWerkz	Annotations or XML file	Plain Java classes	Build-time and runtime	Basic (can benefit from AspectJ merger)	Bytecode manipulation or reflection-based proxy
JBoss AOP	Metadata or XML file	Plain Java classes	Runtime orientation	Good (relatively new feature)	Uses metadata, pre-compiler and/or reflection
DARAC	Uses simple metadata annotations on an AOP Control	Plain Java classes	Runtime orientation (JMX-based)	None	Uses reflection

Summary Table AOP Frameworks Comparison

4. Finally, it returns the object that was returned from the target method invocation

The *handleNotification* method enables runtime weaving by allowing the AOPManager class to be a JMX listener. This notification method is a member of the RemoteNotificationListener interface and serves as the callback mechanism for JMX event listeners. Please refer to Listing 2 for details regarding the AOPManager class.

The last class to discuss is the *AOAspectWrapper* class and its associated *AOAspectWrapperMBean* interface. This class contains all the attributes of an aspect including, for example, its pointcut expression, advice type, and runtime status as far as being enabled or disabled.

To support JMX notifications (to AOPManager), this class also extends the *NotificationBroadcasterSupport* class and implements the *AOAspectWrapperMBean* interface. This custom MBean implements the *getStatus*, *setStatus*, and *showStatus* methods using a string to represent whether the aspect is enabled ("On") or disabled ("Off"). As a result of this JMX-based weaving, users can activate and deactivate aspects via the command-line (refer to Listing 3 for an example).

The DARAC approach to AOP has the following benefits:

1. It takes a KISS (Keep It Simple Stupid) approach to specifying aspects
2. It supports runtime weaving efficiently and elegantly
3. It doesn't dictate anything regarding the nature of an aspect class
4. It doesn't require Java 1.5

And of course, DARAC has a few disadvantages:

1. It's somewhat intrusive with respect to application code requiring objects to be instantiated via a factory method.
2. It doesn't adhere to the same pointcut expression language used by AspectJ, AspectWerkz, and JBoss AOP
3. It doesn't provide any IDE tooling for navigating between application code and aspects/advice

Summary and AOP Futures

What follows is a table summarizing this discussion on AOP frameworks and technologies.

This author believes that AOP will be extremely important to the future of software. However, it's this non-intrusive power that we must consider before casually implementing AOP across the enterprise. For example, current development, debugging, and management/monitoring tools aren't suited to AOP-enabled code. Moreover, most developers and administrators lack experience with AOP. As a result, I recommend that you proceed cautiously before implementing AOP extensively, and choose your AOP framework carefully based on your organizational needs and priorities.

Listing 1: Instantiating Target Object

```
        import controls.POJO;
import controls.POJOInterface;

public class Example
{
  /**
   * @common:control
   */
   private  controls.AspectControl  aomCtl;

   public Example()
   {
POJOInterface pojo = (POJOInterface)
   aomCtl.createProxy(aomCtl, new POJO());

pojo.doSomething("HelloAOP");
   }
}
```

Listing 2: AOPManager Class Summary

```
Refer to attached file named AOPManager.java.
```

Listing 3: Administrative Control Over Weaving (WebLogic 8.1 example)

```
java  weblogic.Admin  –url t3://localhost:7001
-username weblogic  -password weblogic SET
-mbean Aspects:Name=AOProxy.Aspects.TracerAspect.logger
-property Status On
```

NitroX™

PROFESSIONAL TOOLS FOR ECLIPSE™

JSP | Struts | JSF

Professional development for JSP, Struts & JSF
Simultaneous visual/source Struts & JSF configuration
Code completion for all levels
Automatic validation & error checking for all levels
Debugging support for all levels
AppXRay, AppXaminer, AppXnavigator

FREE JSP EDITOR!

download at: www.m7.com/power

M7®

Top Reviews Winner: InfoWorld (8.3), CRN ★★★★★, SDMagazine ★★★★

Setting Up a Java Shop

Building better software faster

by Glen Cordrey

Three times in recent years I've joined an organization that was relatively new to Java development and missing some basic infrastructure elements that I'd relied on in previous development efforts. These elements include utility classes, standards and conventions, and build and quality control tools that help you produce a higher quality product with less risk. If you're involved in a development effort, whether it's new or ongoing, that's lacking any of these elements, you should consider incorporating them into your project infrastructure.

Nuts and Bolts

Some common utility software components should be incorporated into your development efforts as early as possible, because delaying their introduction may result in the proliferation of differing approaches that will need to be reworked later.

Configuration Settings

Access configuration settings via wrappers that hide the settings' underlying storage mechanism. There are numerous places where you can define configuration settings, including properties files, XML files, the database, and via the JDK's preferences package (which on Windows stores preferences to the registry, and to the file system on Unix). If your code uses direct calls to these mechanisms and future needs require that you either change which mechanisms are used or add functionality to those mechanisms, you'll need to make changes everywhere the mechanisms are referenced.

For example, suppose developers store configuration settings in properties files and load and access the settings via calls to the Properties class sprinkled throughout their code. If sometime later you find that changes made to the configuration settings need to be reflected in the application while the application is running, you'll need to change the code that loads those properties to support reloading them. If the settings that need to be reloaded aren't all loaded by the same code – for example, some are UI settings loaded by UI code and others are network settings loaded by network code – you'll need to make the same types of changes in multiple places.

You might also need to change the underlying storage mechanism. For example, a new customer might be database-centric and used to administering configuration settings in database tables and insist that your settings be administered the same way. If you're reusing a code base that has references to the Properties class throughout the code, you'll have to make a lot of changes to accommodate the new customer.

You can roll your own configuration settings classes or harvest them from the Internet. If you can incorporate open source into your product, take a look at the Jakarta Commons Configuration package. If you need or prefer to roll your own, you could start with a simple factory+interface approach as in:

```
public class ConfigFactory {
   public static ConfigFactory getIn-
stance() {…}
   public Config getConfig( ) {…};
}

public interface Config {
   public int getInt( String settingName );
   public long getLong( String setting-
Name );
   …
}
```

You would then implement the Config interface once for each settings repository that you use, as in a ConfigProperties implementation, a ConfigXML implementation, etc.

Logging

If you don't have a logging package in place very early in your coding efforts, you can easily find yourself with a hodgepodge of logging approaches that make error investigation far more difficult than it should be. I've joined a number of large in-progress development efforts where almost every subsystem had its own custom logging package with numerous log files scattered in various directories, and log messages and message timestamps with varying formats. Consequently, one of the first hurdles in investigating a problem becomes determining, and locating, which log files may contain messages related to the problem. Then, if messages of interest are in multiple files, you may need to collate them into a chronological sequence, possibly reconciling different timestamp formats to do so.

Your choice for logging should be between using the JDK logging APIs and Log4j, unless you have specific logging needs that can't be addressed by either package. The March 2005 issue of *JDJ* contained an excellent article, "Log4j vs java.util.logging," by Joe McNamara that can help you in your decision.

If you're developing J2EE applications, an additional factor in your decision should be how easily your log messages can be directed to the application server's logging console and files. Many application server administration UIs have capabilities for displaying and filtering log messages, so if you can direct your messages to the application server's log message store, you can capitalize on those capabilities. In addition, having your log messages automatically collated with the application server's log messages may aid your problem investigation. For example, suppose your application fails because a resource pool in the application server was exhausted, but the error messages reported by your

Glen Cordrey is a Java architect and developer who has worked with Java for eight years. He previously penned articles on J2ME for *JDJ*, and also edited the J2ME section. He works in the Baltimore-Washington area.

glen@oojava.com

application contain insufficient detail to determine the cause. Having your messages in the application server log right after an application server message reports the exhaustion saves you considerable time in understanding the problem.

If you need to support more than one logging mechanism, consider using the Jakarta Commons logging package, which provides a common logging API under which you can plug in JDK logging, Log4j, or a custom logger. However, realize that if you use this common API, you will be unable to access some features of the underlying implementation, as is explained at http://www.qos.ch/logging/thinkAgain.jsp by Ceki Gülcü, a key contributor to Log4j.

Exception Handling

Establish your exception handling approach early to ensure that you have mechanisms in place for the consistent and complete reporting and handling of errors. Decide on the project's philosophy regarding checked and unchecked exceptions – should checked exceptions be wrapped in unchecked exceptions? Your initial reaction might be "Why is this even an issue, since it defeats the point of having checked exceptions?" Well, a number of luminaries in the Java field, such as Bruce Eckel, advocate wrapping checked exceptions in unchecked exceptions (see www.mindview.net/Etc/Discussions/CheckedExceptions). One argument for doing so is that many developers don't really know what to do when a checked exception occurs. Because they're forced (by the compiler) to either catch it or declare it in their method's throws clause, they commit sins such as catching but not rethrowing it, which can make a problem investigation more difficult. (See Joshua Bloch's book *Effective Java* for a more extensive discussion on why consuming exceptions is bad practice.)

Regardless of which approach you subscribe to, consider incorporating default exception handlers into your architecture. The ThreadGroup class has an uncaughtException method that you can use to apply default processing for exceptions that propagate up from any threads in the ThreadGroup. With JDK 1.5, things get even better, as the Thread class has a setUncaughtExceptionHandler method that sets the handler for the thread, and a setDefaultUncaughtExceptionHandler method that sets the exception handler for all threads that don't have their own exception handler.

Shop Layout, Standards, and Procedures

Don't try to reinvent the wheel when it comes to deciding how to organize your project directories and defining development standards, guidelines, and conventions – there are plenty of resources on the Web to which you can refer.

Sun has published directory and naming standards at http://java.sun.com/blueprints/code/projectconventions.html and http://java.sun.com/blueprints/code/namingconventions.html, which should be your preferred starting point unless other considerations are overriding. One such consideration might be your selection of a build tool – for example, Maven (discussed below) has a recommended directory structure.

Sun also has published coding standards, although I find the Sun standards to be rather excessive and prefer fewer standards, with more focus on reducing potential sources of problems. In that light I recommend starting with the AmbySoft standards at http://www.ambysoft.com/javaCodingStandards.html, which also discuss alternative approaches for various items such as parameter naming.

Ant is the de facto standard for building Java applications, but it isn't just for building, as it supports almost all development tasks short of writing code and project management. In addition, many product manufacturers such as application server vendors now provide Ant code for building with, configuring, deploying to, and/or managing their products.

You can incorporate Ant into an automated build and test environment using CruiseControl. The advantages of doing so are explored in Martin Fowler's discussion of Continuous Integration (available from the CruiseControl Web page), and come from the observation that the earlier in the development process you find problems, the cheaper it is to fix them. Continuous Integration helps you find many problems soon after they are inserted into the baseline.

You can also have CruiseControl automatically run various open source tools (discussed below) that identify potential bugs and quality issues in your code. These tools, along with Ant, can log their processing steps and results as XML, which means that their execution can be automatically analyzed and acted upon. For example, CruiseControl can analyze build results and send e-mail reporting on the success or failure of the build and automated tests.

An alternative to the combination of Ant and CruiseControl is Maven, which provides a more comprehensive, project-management perspective. I haven't used Maven and so can't comment on it, but their Web site contains extensive documentation.

Quality Control and Improvement

You can improve the efficiency of your development effort and the quality of your product by incorporating automated tests using JUnit and tools built on top of it such as HttpUnit and Canoo WebTest. HttpUnit provides APIs that you can call to simulate requests from a browser, whereas with WebTest (which uses HttpUnit) you write XML to do the same.

As mentioned earlier, a number of open source products exist to improve the quality of your code. FindBugs and PMD analyze your code to identify possible bugs, including sins such as the previously mentioned consumption of exceptions. JDepend helps you manage dependencies between Java packages, because it's easier to extend, reuse, and maintain packages if the dependencies between packages are well factored. JavaNCSS counts lines of code, number of classes, etc., and also computes cyclomatic complexity numbers (a.k.a. McCabe metrics), which can be used to identify code that is overly complex and should be considered for refactoring.

Most (if not all) of these tools provide Ant targets and plugins for IDEs such as Eclipse. So in addition to running tools such as these as part of your build cycle, you should ensure that developers know about them and routinely run them against their code before checking it in.

Summary

These elements of a project infrastructure are low-hanging fruit – a modest investment of effort to incorporate them into your development efforts early on will provide benefits throughout development, helping you build better software faster.

Is Mobile Java a Reality?

by Michael Yuan

Some casual observations from JavaOne 05

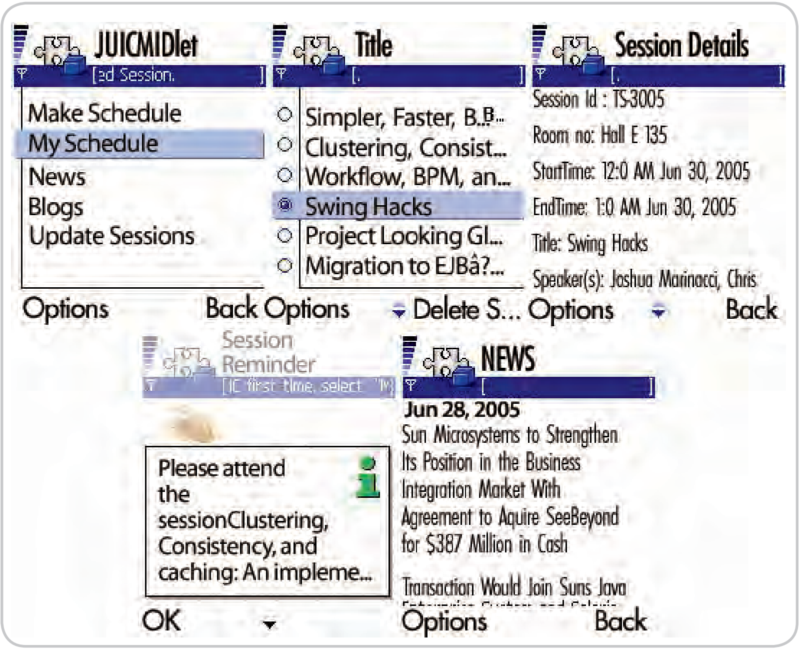
"Java on mobile phones" has been the hottest topic at the JavaOne conference for the past several years. This year was no exception and a large part of the show floor was designated as the "Wireless Village." With tens of billions dollars' worth of Java phones and related services sold every year, Sun and many others are clearly making money. However, most JavaOne attendees I met were enterprise developers. Each year they ask the same questions: "How can I be part of the Java ME success?" "Will mobile Java ever create as many developer opportunities as enterprise Java?" The answers to those questions depend on whether the small and mid-sized businesses can leverage mobile Java to improve productivity and customer satisfaction as they successfully did with enterprise Java.

The answer to how mobile Java can improve productivity lies in integrating mobile phones into enterprise information systems so employees gain real-time knowledge about the business while they're away from their desk. As an example, mobile e-mail on BlackBerry Java devices drives the business in many companies, especially sales and services departments. To illustrate applications beyond mere mobile e-mail, Sun released a mobile phone–based JUICMIDlet (JavaOne User Information Console) application for this year's JavaOne attendees (see Figure 1). JUICMIDlet stored the entire JavaOne session schedule and detailed information for each session on each attendee's phone. This replaced the heavy 200-page book folks used to carry in their backpacks. Right from the phone you can browse sessions based on categories and see their details. If you see an interesting session, you can then add it to your schedule. Just before the scheduled session is due to start, the phone would alert you with flash and sound. In addition, JUICMIDlet downloaded the latest conference news and key JavaOne blogs onto your phone. What's neat is that JUICMIDlet doesn't even require you to have data services on your phone – the application deployment and content updates are all done via Bluetooth on the show floor.

Considering JUICMIDlet's architecture – it already has many of the key elements of enterprise mobile applications: always-on, pervasive, and facilitates information flow. This is similar to the type of app that could be used by folks in field services, sales, or marketing. By building JUICMIDLet, Sun has demonstrated that this type of mobile app is feasible on the vast majority of mobile phones on the market. It also got me thinking about ideas regarding the potential to improve JUICMIDlet using currently available technology.

- Include a more comprehensive set of conference events updates such as show floor vendor presentations, Java.net information, or vendor parties. This could be done with a Web service API on the back end for all vendors to publish their events to.
- Send questions to the speaker in the Q&A session of a 1,000-person standing room session via SMS or voice.
- Feedback surveys from your phone allowing you to fill them out during the session.
- Bluetooth sensors to detect your location in the conference center and provide directions to your next session.
- Borrow an idea from the Nokia Sensor application and allow attendees to publish their own profiles on the phone via Bluetooth. You'll be able to find out who's who in your vicinity during a gathering and strike up a conversation.

Michael Juntao Yuan is a member of *JDJ*'s editorial board. He is the author of three books. His latest book, *Nokia Smartphone Hacks* from O'Reilly, teaches you how to make the most out of your mobile phone. He is also the author of "Enterprise J2ME" – a best-selling book on mobile enterprise application development. Michael has a PhD from the University of Texas at Austin. He currently works for JBoss Inc. You can visit his Web site and blogs at www.MichaelYuan.com.



Session Id : TS-3005
Room no: Hall E 135
StartTime: 12:0 AM Jun 30, 2005
EndTime: 1:0 AM Jun 30, 2005
Title: Swing Hacks
Speaker(s): Joshua Marinacci, Chris

Please attend the sessionClustering, Consistency, and caching: An impleme...

Jun 28, 2005
Sun Microsystems to Strengthen Its Position in the Business Integration Market With Agreement to Aquire SeeBeyond for $387 Million in Cash

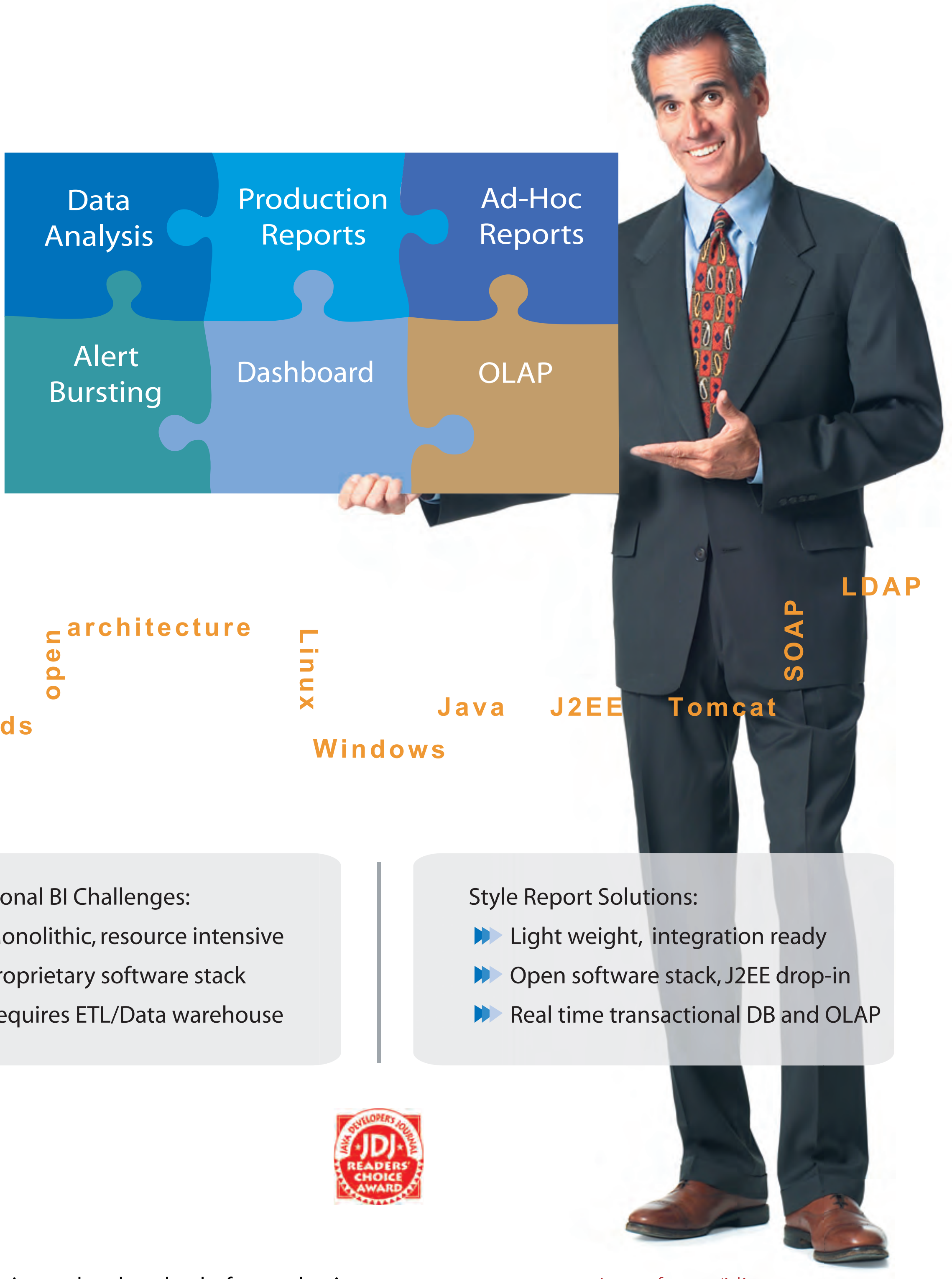
Transaction Would Join Suns Java

Figure 1 JUICMIDlet in action

InetSoft
open standards innovation

Style Report 7

Light Weight, Integration Ready Enterprise Reporting & Analysis



Traditional BI Challenges:

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

Style Report Solutions:

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

JDJ JAVA DEVELOPER'S JOURNAL READERS' CHOICE AWARD

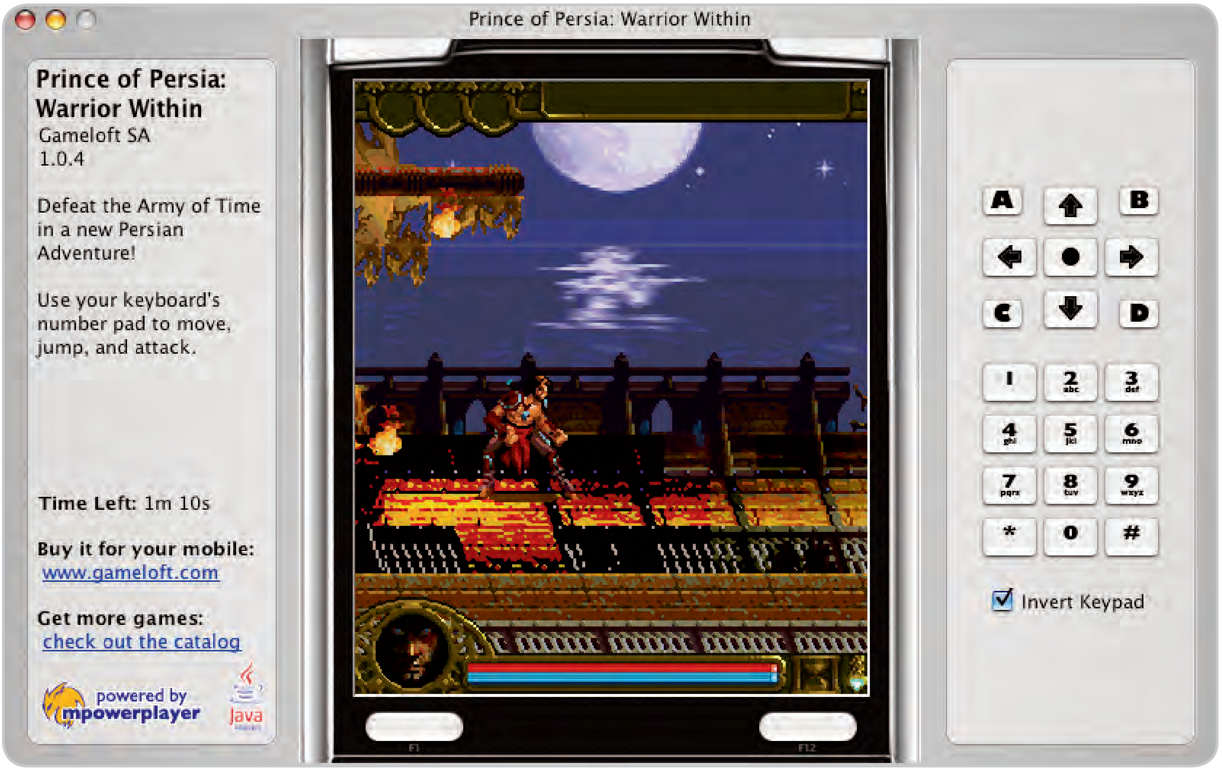
For more information and to download a free evaluation copy .... www.inetsoft.com/jdj

"The answer to how Java can improve productivity lies in integrating mobile phones into enterprise information systems"

In addition to enterprise applications, mobile entertainment consumer apps are another usage of mobile Java. This is already a multi-billion dollar business and still growing. A significant hurdle, however, to the adoption of mobile Java games is that most consumers are unaware of what games their phones are capable of running. Also, the business model relies on consumers buying cheap games often. Offering trial downloads for every $5 game is not economic for game developers. In this year's JavaOne Wireless Village I found a perfect solution from mpowerplayer (www.mpowerplayer.com).

The mpowerplayer product is a Java mobile phone simulator for regular computers that runs on the standard Java environment (JDK 1.4) over Java Web Start. It allows consumers to run mobile Java games on their PC or Mac through a regular Web browser. Consumers can learn about the graphic quality, user experience, and game play of the game on their PC before making a purchase. The developers have very little work involved to repackage the existing mobile Java game for mpowerplayer. The Texas Hold'Em poker contest on the show floor was how many JavaOne attendees got a taste of the quality of mobile games firsthand. Many users I talked to felt that the graphics quality was good enough even for playing on the PC screen, let alone a mobile phone. I strongly urge you to play some free trial games from mpowerplayer.com on your PC and see how far mobile phone games have come! Figure 2 shows "Prince of Persia" running on the mpowerplayer.

JUICMIDlet and mpowerplayer are just two examples of interesting mobile Java applications at JavaOne 05. Nokia and Motorola also had an array of exciting new Java devices on display in their booths that I can't wait to get my hands on! Nokia announced support for the Java ME CDC (Connected Device Configuration) profile on their popular Series 60 devices (25 million units shipped). Since the CDC is close to Java SE and supports much more API than MIDP, it opens up fresh opportunities for both mobile developers and users, especially for the enterprise applications. Sun also released support for the Nokia SNAP API for multi-player mobile game servers in the Wireless Toolkit |package. This is an important step for mobile game developers to leverage Nokia's advanced mobile game infrastructure and player communities. All in all, JavaOne is definitely an exciting place to be for mobile Java developers.



Prince of Persia: Warrior Within
Gameloft SA
1.0.4

Defeat the Army of Time in a new Persian Adventure!

Use your keyboard's number pad to move, jump, and attack.

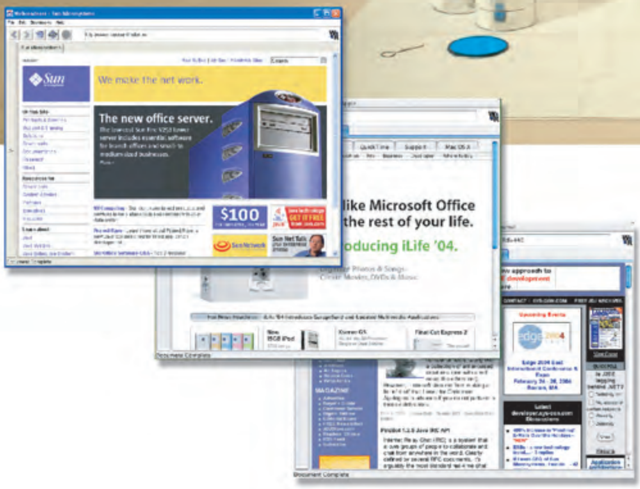
Time Left: 1m 10s

Buy it for your mobile: www.gameloft.com

Get more games: check out the catalog

Figure 2 Prince of Persia in mpowerplayer

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.

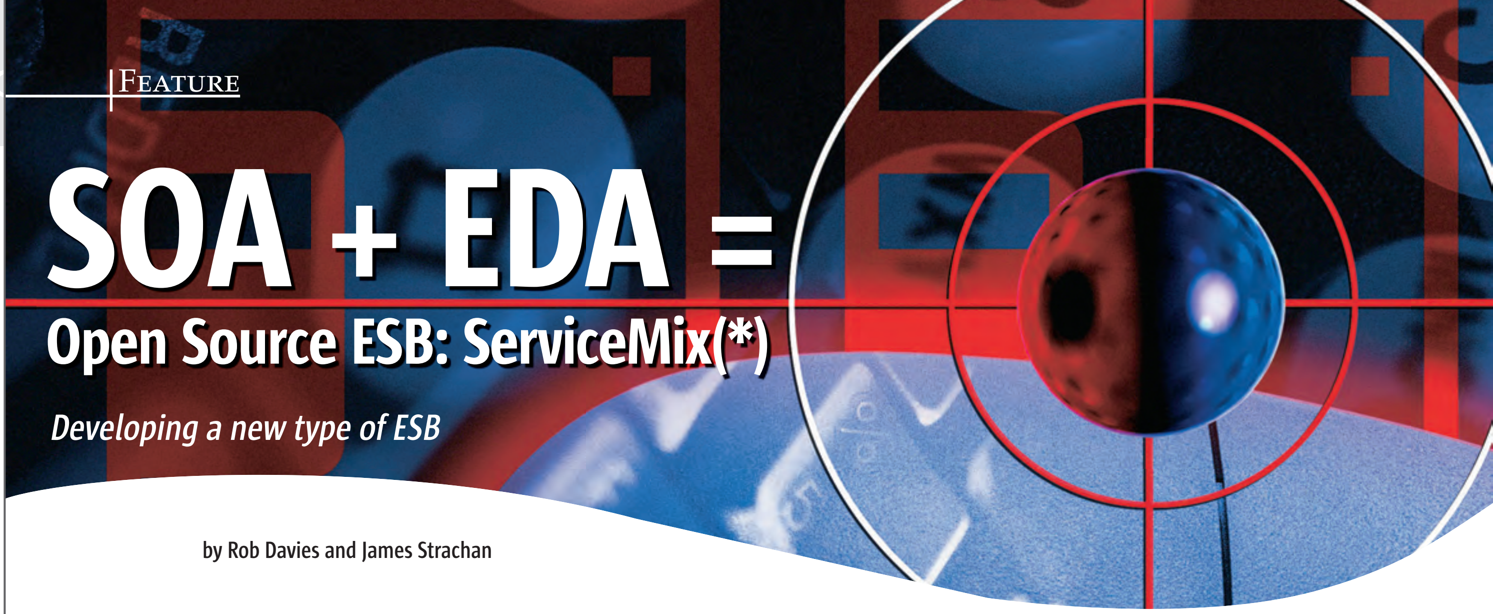
Why WebRenderer?

- 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



SOA + EDA = Open Source ESB: ServiceMix(*)

Developing a new type of ESB

by Rob Davies and James Strachan

Today's enterprise applications are distributed by design. For applications to interact with one another over networks optimally, they require Service Oriented and Event Driven Architectures made up of loosely federated business resources, that interact by exchanging requests (for data delivery and integration, as well as for services) and that can handle streams of diverse business processes in real-time. To support large-scale, enterprise integration, organizations need to adopt strategies that rationalize the infrastructure for integration based on the requirements of business/IT organization itself. The only successful integration efforts are those that provide agile, pervasive and low cost solutions in order to cater to today's diverse deployment environments, while fully leveraging available standards.

Enterprise Service Bus (ESB), which can be defined as middleware that brings together both integration technologies and runtime services to make business services widely available for reuse, offers the best solution for meeting today's enterprise application integration challenges by providing a software infrastructure that enables SOA. However, there are currently a number of different vendors that provide ESB solutions, some of which focus purely on SOAP/HTTP and others who provide multi-protocol capabilities. Because these vendors span the horizon from big enterprise generalists (app servers), to mid-tier enterprise integration providers, all the way to smaller, ESB/integration specific-providers – there doesn't seem to be an established consensus regarding the key requirements for an ESB.

As application architects, we have often thought about what requirements would define an ESB designed specifically to cater to the needs of an agile, enterprise integration model. In building for these specific requirements, we realized that we actually needed to develop a new type of ESB – hence the ServiceMix project.

James Strachan has been writing enterprise software for over 20 years now from C, C++, Smalltalk, Java, and more recently Groovy. He is an active open source developer and one of the founders of the Apache Geronimo project (Apache's J2EE container) and the Groovy programming language as well as other projects like ActiveMQ, ActiveCluster, ActiveSpaces, jelly, dom4j and jaxen and works on a variety of other open source projects like Spring, PicoContainer, Axion, drools, Maven, and Jakarta Commons.

jastrachan@mac.com

Characteristics of an Agile ESB

The main criteria we were looking for in our ESB are as follows:

- ***Standards based*** **-** While standards-based support is marketed by many ESB vendors, the support is provided externally, requiring developers to work with proprietary APIs when directly interacting with internal APIs. ServiceMix was designed with the requirement to eliminate product API lock-in, by being built from the ground up to support the Java Business Integration specification (JSR 208). Our agile ESB needs to use JBI as a first class citizen, but also support POJO deployment for ease of use and testing.
- ***Flexible*** **-** Another characteristic of an agile ESB is the flexibility with which it can be deployed within enterprise application integration framework: standalone, embedded in an application component, or as part of the services supported by an application server. This allows for component re-use throughout the enterprise. For example, the binding for a real-time data feed might be aggregated as a web-service running within an application server, or streamed directly into a fat client on a traders desk. An agile ESB should be able to run both types of configurations seamlessly.

  To provide rapid prototyping, an agile ESB should support both scripting languages and embedded rule engines, allowing business processes to be modeled and deployed quickly.

  Some have argued that integration functionality is best place at the edges of the network. Others prefer a logical ESB server to be separate from the edges – to keep the edges simple and lightweight. Both approaches have their strengths and weaknesses – so we wanted an ESB that is simple and lightweight to deploy into any JVM or into a web server or a full Java EE server – reusing all the available facilities in which it is deployed.
- ***Reliable*** **-** Our ESB needs to handle network outages and system failures and to be able to reroute message flows and requests to circumvent failures.
- ***Breadth of Connectivity*** **-** An agile ESB must support both two way reliable Web-services and Message Oriented-Middleware and needs to co-operate seamlessly with EIS and custom components, such as batch files.

  In addition we want support for the various new WS-* standards to do with connectivity like WS-Notification, WS-DistributedManagement and WS-ReliableMessaging.

We also wanted our agile ESB to be vendor independent and open source, to promote user control of source code and direction. An added benefit of this is not only the zero purchase cost, but the total cost of ownership will be reduced where users are actively contributing and maintaining our ESB.

We rapidly came to the conclusion, that as there was no single product that would adequately meet our needs, we'd have just go a head and build one!

What Is JBI?

There has been a fair amount of buzz about JBI and there is some confusion over what JBI (JSR 208) is.

JBI is a simple API to a Normalized Message Service and Router along with a component and management model for deploying integration services such as routing engines, BPEL engines, rule systems, transformation engines etc.

JBI provides a logical XML messaging network which maps well to web services, HTTP, email and JMS/MOM while easily adapting to legacy systems, binary transports and RPC systems like EJB and CORBA. Think of it as the next logical abstraction above JMS, with support for different message exchanges (one way, request response etc).

The *binding components* deal with all the plumbing and protocol stuff, then *service engine components* work on a logical XML layer, providing content based routing, orchestration, rules, transformations and custom enrichment etc.

So BPEL engines no longer need to deal with all of the possible protocols, transports and wire formats; they can just delegate to JBI for the physical routing to service endpoints. Similarly content based routers, rules engines, transformation engines can sit on the JBI bus and do their thing. JBI is looking like being a great API for integration component developers.

Many application developers will still end up writing POJO services and dropping them into their container and exposing them as web services - so often they won't need to use the JBI APIs directly; but for integration vendors and open source integration projects, JBI provides a way for us to all work together at the ESB level and to reuse integration containers, components and tooling.

ServiceMix

ServiceMix is an open source (Apache licensed) Enterprise Service Bus which is compliant with the Java Business Specification (JBI), JSR 208.

ServiceMix already provides JBI support for Apache Geronimo, the first application server to provide this feature.

The ability to use a standard for the deployment and management of integration components is essential if integration architects, developers and component vendors are not to be tied to the proprietary API's that have existed to date when deploying within an ESB.

However JBI is quite strict in the way Components are installed and services deployed. JBI mandates that each Component must be installed from an archive, containing a well defined XML descriptor file, and that the archive must be unpacked on the local file system and the installation component provided with it's own working directory, if it requires it.

This strict contract does not lend itself well to ease of use, quick development or testing - and negates some deployment scenarios where you would to use an ESB as a lightweight container embedded in an application fragment or library.

Hence ServiceMix has taken the approach that while JBI is a first class citizen (it's a JBI compliant container) and internally all message flows are routed using JBI constructs (like the Normalized Message Service), components can be additionally deployed as POJOs. ServiceMix is also tightly integrated with the Spring framework, allowing Spring to deploy integration components for you if you so wish within a regular Spring context.

Reliable

ServiceMix is designed to easily support simple and automatic distribution of components and message flows. Internally, ServiceMix distributes events using a plug-able message routing architecture, called a Flow.

By default, ServiceMix supports three Flow types for message routing:

- ***STP*** **-** straight-through interactions, components are interacted with directly. This is ideal for embedded or light weight deployment.
- ***SEDA*** (Staged Event Driven Architecture) for scalable message routing
- ***Clustered*** **-** Components seamlessly register themselves with other ServiceMix instances in a cluster, allowing for seamless distributed event propagation

Breadth of Connectivity

ServiceMix can handle any JBI standard component; so components from other open source projects such as PXE from FiveSight or components from the Celtix project should just drop right in.

In addition, ServiceMix comes with a whole raft of reusable JBI components

- SAAJ for working with Soap With Attachments providers such as Apache Axis
- WSIF for working with any Web Service Invocation Framework implementation
- ActiveSOAP and XFire support to provide clean integration with new lightweight SOAP stacks
- Scripting support with JSR 223 or Groovy to allow powerful and agile integration
- HTTP, JMS, email and Jabber transports to provide a general message bus
- JCA support for fast and efficient processing of messaging resources like JMS with connection, session and thread pooling as well as efficient parallel processing and transaction & exception handling
- Quartz and JCA WorkManager support for enterprise timer integration
- Caching support with JCache integration to allow any service invocation to be cached among a cluster based on some correlation or request key (using XPath or Java code to extract the key)
- XSLT support to allow transformations to be used in pipelines
- Reflection, Spring and Mule support for clean POJO integration
- SQL support with Oracles XSQL tool to provide CRUD operations inside message flows

JBI Client API

To make it simpler to use ServiceMix for developers, we've created a JBI Client API which makes it easy to work with any JBI container or any available JBI component.

Using the JBI Interfaces

The following ServiceMix methods provide some helper methods for easier use of the JBI APIs

Sending Messages One Way

This example uses a specific service to make an invocation function call:

```
InOnly exchange = client.createInOnlyExchange();
NormalizedMessage message = exchange.getInMessage();
message.setProperty("name", "James");
message.setContent(new StreamSource(new StringReader("<hello>
```

Rob Davies has developed enterprise software products for over 20 years and is currently working for Exist Engineering, which provides software solutions to the global market using open source. Before joining Exist, Rob was the CTO and founder of SpiritSoft, an enterprise messaging company. Rob is currently developing the JBI container for ServiceMix, an open source ESB.

rajdavies@exist.com

```
world</hello>")));

// lets use a specific service to dispatch to
QName service = new QName("http://servicemix.org/cheese/",
                          "receiver");
exchange.setService(service);
client.send(exchange);
```

In this example, we assume that the JBI container will have setup a default routing connection for our client, so we don't have to worry about specifying the endpoint.

```
InOnly exchange = client.createInOnlyExchange();

NormalizedMessage message = exchange.getInMessage();
message.setProperty("name", "James");
message.setContent(new StreamSource(new StringReader
                  ("<hello>world</hello>")));

client.send(exchange);
```

Invoking Services with InOut Exchanges

```
InOut exchange = client.createInOutExchange();

NormalizedMessage inMessage = exchange.getInMessage();
inMessage.setProperty("name", "James");
inMessage.setContent(new StreamSource(new StringReader
                    ("<hello>world</hello>")));

// optionally specify the endpoint
exchange.setService(service);

client.sendSync(exchange);
NormalizedMessage outMessage = exchange.getOutMessage();
```

Using the POJO Methods

Following are a few helper POJO-based methods, provided to allow you to use ServiceMix with regular POJOs to hide some of the JBI's XML marshalling details.

This allows you to use a plugable Marshaler to map your POJOs to JAXP Sources.

Sending Messages

This example uses a specific service to make an invocation call:

```
Map properties = new HashMap();
properties.put("name", "James");
// lets use a specific service to route to
QName service = new QName("http://servicemix.org/cheese/",
                         "receiver");
EndpointResolver resolver = client.createResolverForService
                           (service);
client.send(resolver, null, properties, "<hello>world</hello>");
```

In the next example, we assume that the JBI container will have set up a default routing connection for our client, so there is no requirement to specify the endpoint.

```
Map properties = new HashMap();
properties.put("name", "James");
client.send(null, null, properties, "<hello>world</hello>");
```

Invoking Services with InOut

```
// optional endpoint resolution
EndpointResolver resolver = client.createResolverForService
                            (service);
Map properties = new HashMap();
properties.put("name", "James");

Object response = client.request(resolver, null, properties,
"<hello>world</hello>");
```

Example Using JMS and XSLT

Here's a quick example to show you some of the ServiceMix integration capabilities in action. We consume messages using JCA, then transform them with XSLT and send them to a new destination using JMS.

Let's set up a JBI component to consume from JCA using JMS:

```
<component id="myJmsReceiver" service="foo:myJmsReceiver"
 class="org.servicemix.components.jms.JmsInUsingJCABinding"
  destinationService="foo:transformer">

  <property name="jcaContainer" ref="activeJcaContainer"/>

  <property name="activationSpec">
    <bean class="org.activemq.ra.ActiveMQActivationSpec">
      <property name="destination" value="test.org.servicemix.
                                              example.jca/>
   <property name="destinationType" value ="javax.jms.Topic"/>
    </bean>
  </property>
</component>
```

Let's transform the message

```
<component id="transformer" service="foo:trans
 former" class="org.servicemix.components.xslt.XsltComponent"
  destinationService="foo:transformedSender">
  <property name="xsltResource" value="classpath:org/servicemix/
 components/xslt/transform.xsl"/>
</component>
```

Now let's send the message using the Spring JmsTemplate

```
<component id="myJmsSender" service="foo:myJmsSender" class="org.
 servicemix.components.jms.JmsSenderComponent">
  <property name="template">
    <bean class="org.springframework.jms.core.JmsTemplate">
      <property name="connectionFactory">
        <ref local="jmsFactory"/>
      </property>
      <property name="defaultDestinationName" value="test.org.
                    servicemix.components.xslt.source"/>
      <property name="pubSubDomain" value="true"/>
    </bean>
  </property>
</component>
```

Ensure Interoperability.

Validate functionality.

Eliminate security vulnerabilities.

Test performance & scalability.

Confirm compliance.

Collaborate thru reuse.

Ensure Secure, Reliable Compliant Web Services

PARASOFT®
SOA Test™

As enterprises adopt Service Oriented Architectures (SOA) to deliver business critical data, ensuring the functionality and performance of Web services becomes crucial. Complex web services implementations require the means to thoroughly validate and test them to assure they are truly production ready.

Parasoft SOA Test is a comprehensive, automated tool suite for testing web services and complex Service Oriented Architecture (SOA) solutions to ensure they meet the reliability, security and performance demands of your business. SOA Test provides a total and holistic testing strategy for your SOA implementations including automated unit testing, graphical scenario testing, scriptless performance/load testing, security penetration testing, standards validation, message authentication, and more.

If you are building serious web services, you need SOA Test.

Download a copy of SOA Test for a free evaluation today at www.parasoft.com/SOA_WSJ

Parasoft SOA Test clients include: Yahoo!, Sabre Holdings, Lexis Nexis, IBM, Cisco & more.

PARASOFT®
We make software work.™

Automated Software Error Prevention™

Scripting Support

ServiceMix also supports scripting languages through JSR 223 – Scripting for Java. Here are some examples using Groovy:

Before we go into detail of how you can work with JBI and Groovy in ServiceMix, lets show a simple hello world kinda example.

```
<component id="myServiceUsingXMLText" service="foo:myServi-
ceUsingXMLText" endpoint="myServiceUsingXMLText" class="org.
servicemix.components.groovy.GroovyComponent">
        <property name="scriptText">
          <value>
            <![CDATA[

// lets output some message properties
outMessage.properties = [foo:"hello", someList:[1, 2, 3]]

// lets output some non-xml body
outMessage.bodyText = """
<hello>
  <world person="$inMessage.properties.name"
location="London"/>
</hello>
"""
            ]]>
          </value>
        </property>
      </component>
```

As you can see the component is configured with a piece of Groovy to execute when the service is invoked.

Now we'll go through the various options which are available when working with JBI and Groovy in ServiceMix.

Maintaining State Across Requests

It's often handy to keep track of state across requests. There is a variable called 'bindings' which you can use to maintain state; here's the groovy...

```
if (bindings.counter == null) {
    bindings.counter = 1
}
else {
    ++bindings.counter
}

def date = new Date()

outMessage.bodyText = "<response counter='$bindings.counter'
date='$date'></response>"
```

Working with JBI Properties

In ServiceMix you can access the JBI message properties as a Map and work natively with it in Groovy using various mechanisms. e.g.

```
// lets output some message properties
outMessage.properties.foo = "hello"
outMessage.properties.someList = [1, 2, 3]
```

or use an intermediate object if you've lots of properties to set

```
def props = outMessage.properties
props.foo = "hello"
props.someList = [1, 2, 3]
```

or just use the native Map/property syntax

```
outMessage.properties = [foo:"hello", someList:[1, 2, 3]]
```

Generating Output

Groovy provides various mechanism for generating the output (whether it is the result of a service or a transformation). Which mechnism you use depends on your use case and personal preference.

String Templates

You can use Groovy string templates to output XML, which is a nice, simple way to generate blocks of XML with dynamic content:

```
outMessage.bodyText = """
<hello>
  <world person="$inMessage.properties.name"/>
</hello>
"""
```

Notice the user above of the input messages's 'name' property, which is equivalent to the expression

```
inMessage.getProperty("name")
```

POJO Return Values

You can return a POJO as the body of a message - which other components can either transform or the default Marshaler will figure out the right thing to do.

```
// lets output the body as a POJO
outMessage.body = [3, 2, 1]
```

Using Groovy Markup

Groovy supports a simple and concise markup mechanism which can be used to programatically generate some XML markup (either DOM, SAX or any other XML model) while retaining the full power of Groovy within the control flow of the markup.

```
// lets output some XML using GroovyMarkup
outMessage.body = builder.hello(version:1.2) {
  world(person:inMessage.properties.name, location:'London')
}
```

Conclusion

If you are interested in SOA, EDA and integration please take a look at the ServiceMix project and see if it can help you. We welcome contributions!

Resource

ServiceMix: http://servicemix.org/

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

Think JReport.

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

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

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

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

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

When it comes to embedded Java reporting, JReport is the perfect fit. Download a FREE copy of JReport today at www.jinfonet.com/jp6.

JReport
JINFONET SOFTWARE

Calvin Austin
Core and Internals Editor

C#: Is the Party Over?

One of my tasks at Sun was to keep abreast of the technologies in the marketplace that competed with Java. At certain points in the release we would summarize where we were compared to other technologies and, if necessary, focus on areas where we could improve.

The biggest unknown at the start of my last project was C# and .NET. I heard through the grapevine that a project from Microsoft, known as "Cool," was on its way, a project that was the forerunner to C#. However, it was less than a year before the Java 5 project started that both those technologies were publicly announced.

Five years later what do we see? The .NET platform has been under constant development, often too fast for many corporate users to adopt. There has been a 1.0, 1.1, and 2.0, each which could be counted as a significant version in their own right. Following the churn of the .NET SDK, the Visual Studio product has required its own aggressive update schedule, although when comparing feature lists, C# is not singled out for any special attention on Visual Studio's Web pages. Looking at the forums, Visual C++ and Visual Basic and not C# attract the lion's share of the forum attention. In addition, the underground community site, gotdotnet.org, has undergone significant site and management changes. Given that C# hasn't necessarily been the instant success that many thought it would have been, it hasn't been for lack of trying. The MSDN site has adopted many of the best practices used on other developer Web sites. You can now read and vote on C# bugs and submit suggestions among other community-building initiatives. The C#, C++, and C compilers are now free, although not obviously as optimized as the professional edition. While C# has gained some traction in those years, why didn't it make the grade?

Java Didn't Stand Still

The first reason I can attribute to C#'s struggle is that the Java platform did not stand still. Many of the benefits that the Java platform delivered were not solved by moving to C#, the most significant difference being OS independence. While C# was in rapid release mode, the Java platform was able to fine-tune the language and at the same time invest heavily in stability and scalability. At an application level, the differences are even more marked. Deploying a .NET service leaves a company a small choice of application servers and OS versions. The reverse is true of Java and J2EE, where there were almost too many J2EE application servers to choose from. The market has now moved to an open source J2EE application server model, which brings me to my next point: the open source movement.

Open Source Changes Everything

The momentum of the open source movement has often been documented as being a threat to the proprietary software market, yet at the same time analysts have questioned the validity of a never-ending supply of free labor. The truth is somewhere in between. While developers had to get budget approval for MSDN licenses, their Java colleagues were able to deploy a system for free. Now with the advent of a new crop of open source J2EE application servers to follow JBoss, the justification for a team to spend thousands of dollars on basic development tools becomes harder, especially if it means a choice between deciding on a new laptop and a renewal of your existing desktop tools.

The growth of open source Java hasn't stopped there. You only have to look at Hibernate, the Spring Framework, and Struts/Shale to see that developers can work together to solve their own problems. Being open source doesn't necessarily mean those developers have to work for free; however; it does provide a way for individuals and companies to work together without being restricted by working group policies or internal company politics.

The Mono project, which aims to provide an open source implementation of C# and .NET, has also been around for four years now and is now part of Novell. Providing the compiler is only part of the challenge. The .NET platform uses many Windows services that until Mono started didn't even exist on Linux. Microsoft has awoken to the open source movement; how much they will help Mono is yet to be seen. Mono today is still a development project much as .NET is still looking for full traction.

Conclusion

Is the C# party over? If the plan of C# was to slow the defection of Visual C++ developers to Java, then it was certainly better than nothing. The long-term savings for Microsoft in sharing a CLR between projects was more than worth the initial effort. However, C# is still not the de facto choice for Web site or enterprise development and other languages such as Python and PHP, which are bringing in a new generation of developers who don't have a need to migrate Visual C++ applications. C# isn't going anywhere soon but its best days may be behind it.

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

calvin.austin@sys-con.com

LinuxWorld® CONFERENCE & EXPO

CONFERENCE:
AUGUST 8 – 11, 2005
EXPO:
AUGUST 9 – 11, 2005
Moscone Center West
San Francisco, CA

WHERE open minds MEET > >

explore > > analyze > > gain > >

LinuxWorld Conference & Expo is the world's leading and most comprehensive event focusing on Linux and Open Source solutions. At LinuxWorld, see and learn how to best leverage the technology for your organization.

> **Explore** your options on the exhibit hall floor, which features the world's leading hardware and software vendors.

> **Analyze** the latest Linux and Open Source technology and discover how companies across the globe can show you how to achieve higher profits and increase productivity.

> **Gain** knowledge about best practices and solutions by attending LinuxWorld's outstanding educational program.

It's the Linux & Open Source event you can't afford to miss!

linuxworldexpo.com

> Register Online With **Priority Code: D0106**

PLATINUM SPONSORS

Computer Associates® hp invent IBM Novell ORACLE Sun microsystems

IDG WORLD EXPO

Observed Benefits of the Unified Modeling Language

by Duncan Jack

The UML is designed to serve you

Over the past 12 months, I have observed significant benefits using the Unified Modeling Language (UML) when developing Rich Internet Applications using Macromedia's Flash Platform and JRun (Java application server).

This article first discusses what the UML is, then lists some of the main diagram types. It highlights how these diagrams can be used and draws attention to some of the benefits I've observed when using them. It concludes with a list of resources.

What Is the Unified Modeling Language

To understand the essence of the UML, consider the elements of its name:

- ***Unified:*** The result of unifying three leading approaches to system modelling in the 1990s
- ***Modeling:*** concerned with the simplified representation of system structure and behavior
- ***Language:*** A language, not a methodology

The UML provides a language-neutral, tool-supported, well-documented standard for modeling systems such as Web applications. It enables system requirements, structure, and behavior to be succinctly captured and effectively communicated.

At the time of writing this article, the UML 2.0 Specification is going through final editing, although you'll find that many books and tools support at least a subset of this specification. A draft version of the specification is available on the Object Management Group's UML Web site. Helpful note – don't try and learn the UML from this document, but it can make interesting reading!

The UML is not a methodology. This point is important. Some people think that you have to use every diagram type to model every aspect of system behavior all the time as part of a complex, cumbersome approach. Not at all. Simply make intelligent choices about what works for you. The UML is designed to serve you, not the other way around.

To illustrate this point, consider the Java programming language. Java is a language, not a methodology. To derive full benefit from your use of Java, you adopt an effective methodology. You may adapt your approach on different projects. You use a subset of Java to build an application. You don't try and use every feature of the language in every application you build.

In the same way, blend the UML into the successful methodology you already use.

Main Diagram Types

Essentially, when you use the UML, you draw diagrams and add notations to them. You may draw a UML diagram by hand on the back of a menu over lunch with a client or on a whiteboard. Equally, you may use a tool such as MagicDraw UML.

There are two main categories of UML diagrams defined by the UML 2.0 Specification:

- ***Structure Diagrams (six):*** Concerned with modeling static structure (architecture)
- ***Behavior Diagrams (seven):*** Concerned with modeling dynamic behavior

I have found the following diagrams most useful since I began using the UML 12 months ago:

- Use Case Diagram (behavior)
- Activity Diagram (behavior)
- Class Diagram (structure)
- Sequence Diagram (behavior)

Using these four diagrams in sequence has been very effective, so I will address each in turn.

Use Case Diagram

Use case diagrams help to define the requirements of a system from the user's perspective – what they want to achieve when using the system.

The use case diagram is deceptively simple yet incredibly powerful. Notes are added to the diagram, and may of course be supplemented by other documents where appropriate. This is exactly what architects and engineers in other disciplines do too, of course – use blueprints and drawings.

The user may be a human object or software object (if you are developing

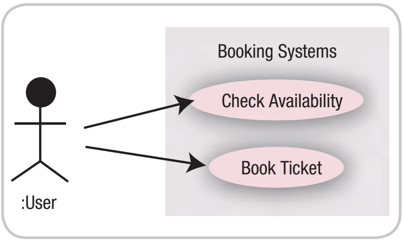


Figure 1 User Case Diagram

Duncan Jack is the founder of Scottish Java (www.scottishjava.com), a brand new Java community. He is also involved in research to develop systems that will assist in measuring, monitoring, and managing organizational quality with the aim of improving business performance. This research leverages his 20 years of successful commercial experience in civil engineering, financial services, and the U.S. oilfield.

duncan@scottishjava.com

"The UML provides a language-neutral, tool-supported, well-documented standard for modeling systems such as Web applications"

"Java is a language, not a methodology. To derive full benefit from your use of Java, you adopt an effective methodology"

a Web service in Java for example). The basic syntax is very simple (see Figure 1).

As you can see from the diagram in Figure 1, the system is required to let a user check availability and book a ticket. Notice that the diagram does not go into the detail of how this will be accomplished. It helps them focus on desired outcomes and not the process. For me, that's the power of use case diagrams – focusing minds and drawing out detail. Of course, it's important to remember that clients may:

- Not know exactly what they want or need.
- Be reporting to a boss who has given them unclear, incorrect, and incomplete requirements.
- Be part of a wider team among which requirements are fragmented.
- Forget or contradict their own requirements.

I have noticed a number of business benefits when using use case diagrams. It's the simplicity of the diagram and the practice of going through the process with a client that really pays off. I've noticed that these diagrams help to:

- Discover what clients actually want and need
- Draw in other stakeholders (the boss, co-workers, etc.) to the requirements gathering process on an ongoing basis
- Identify any correct and contradiction in requirements

I was recently involved in a project to build a Rich Internet Application for a business run by three extremely capable directors in their 50s. They found the use case diagram indispensable. At every meeting, the first thing we would do was review it, to confirm that all requirements had been captured and were fully up to date.

As additional requirements were identified, these were either added into the current version or added to a list for future discussion. Either way, it was up to the clients to decide. The use case diagram was a living, breathing document that provided an ideal way to ensure that the interface with the clients remained cohesive.

Letting them each have a copy of the diagram that they could mark up and use in their own internal meetings proved to be a very effective way to draw everybody in and ensure we built the right system.

We became a natural extension of their business; they became a natural extension of our project team. As a result, meetings were more productive, a better application was delivered more quickly, and business was stored up for the future. In addition, our approach helped us to differentiate ourselves from our competition and ensure a strong ongoing relationship with the clients.

Activity Diagram

Once the requirements of a system from the users' perspective have been defined, activity diagrams help to define how this user experience will be achieved.

Activity diagrams are also extremely powerful. They are well suited to fleshing out the details of a use case by modeling the detailed interaction between a user and a system or screen. Activity diagrams are used to model:

- Business processes.
- Flow of control in an executing program.
- Details of a method.

They are a close relative of the traditional flowchart (see Figure 2). As you identify and diagram the different activities, you'll naturally see a pattern of objects emerge to which the different activities can be assigned. You can use the swim lanes to assign responsibilities to different objects – whether those objects are people or software.

Class Diagram

Class diagrams are used to model the classes of objects in a system (people and software). In the context of this article, the software building blocks are likely to be Java classes.

Think of a class – it has properties (attributes and associations) and methods and can be represented as shown in Figure 3.

The Order class has an orders property, which is an array of order items, each one represented by an OrderItem object. Although this could simply have been shown as an attribute inside the class, it's often more meaningful to represent such a property using an association as above.

On this point, I found Martin Fowler's excellent book, *UML Distilled*, particularly helpful. I highly recommend it. He goes into class diagrams in some detail and wisely splits his coverage into two chapters, focusing the first chapter on the essentials.

Class diagrams seem to follow so naturally from activity diagrams; the activities identified often may neatly correspond to methods in a class diagram, which helps save time and increase productivity.

There is, of course, no requirement to identify every property or method on a class in a class diagram. You may choose to show only public methods for example. Equally, you don't have to show all classes and relationships between them. Again, use what works for you.

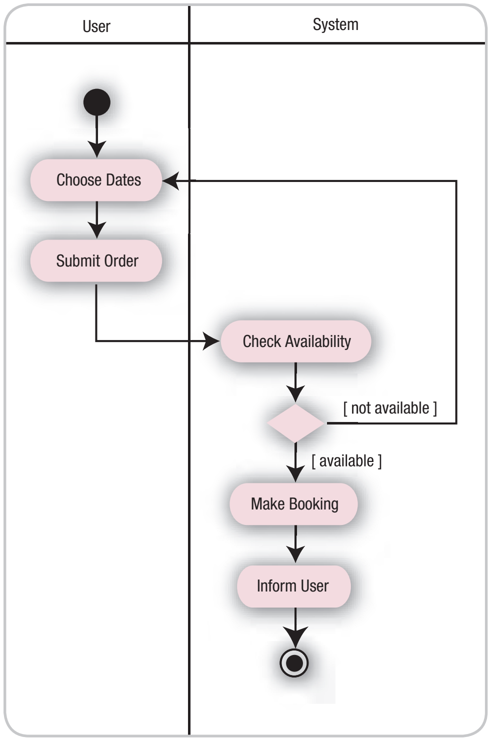


Figure 2 Activity Diagram

"For me, the biggest benefit of the UML has been the sequence diagram"

Class diagrams really help when architecting the system and seem to give the design "room to breathe." It seems that if the design is elegant, the implementation is elegant too. If the implementation is elegant, it can be more pleasurable and cost-effective to evolve and maintain on an ongoing basis once the system has been put into production.

In Ian Sommerville's definitive work, *Software Engineering*, he cites (and qualifies) research that suggests up to 90% of software costs are evolution costs. Looking at this another way, if all you build for a client is the initial implementation of a system, you may only be getting as little as 10% of the revenue stream that you would otherwise get from that client over the lifetime of the system. Of course, these figures will vary significantly, but it's an interesting thought.

If you build a system that can be evolved elegantly and cost effectively, you're more likely to keep the relationship with the client, give them a better service, and make more money. Class diagrams are great for organizing where functionality will go, and for helping to select consistent and meaningful property and method names.

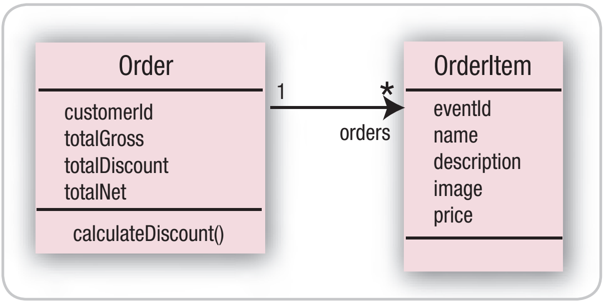


Figure 3 Class Diagram

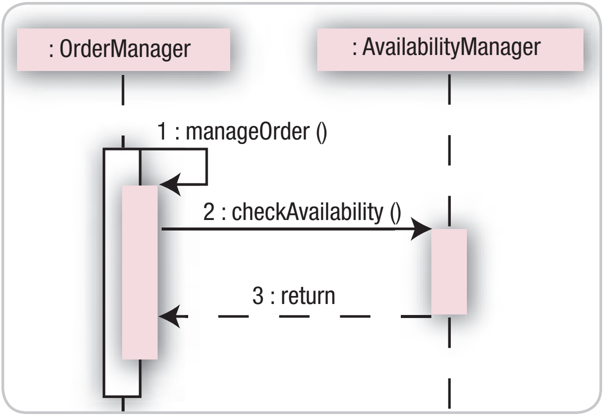


Figure 4 Sequence Diagram

Sequence Diagram

Have you ever developed an application and then had to come back and modify it six months later and tried to work out how on earth you did it? Well, the UML sequence diagram may be able to help you.

A sequence diagram models the sequence of interactions between objects. In some ways, it a close cousin of an activity diagram, yet focused more on the behavior of software objects on a timeline (see Figure 4).

Sequence diagrams are great for thinking through a design, illustrating an idea, and also getting back up to speed when changes need to be made six months or so after the system has gone into production. The design stands out so clearly.

Prior to finding out about the UML, I used my own non-standard diagrams. For me, the biggest single benefit of the UML has been the sequence diagram.

In the UML modeling tool I use, MagicDraw UML, I typically have a class diagram open at the same time as a sequence diagram. As I work on the design and identify additional methods, I add these to the appropriate class. These methods are then immediately available for me in the sequence diagram. As a result, it's much easier to create an elegant design and enhance productivity. Another powerful feature of MagicDraw UML is that it enables me to generate all the framework code in Java from the UML model at the click of a button (equally, I can reverse engineer a sequence diagram from Java code).

It also ensures that the design is the documentation, which ensures that the documentation is done as the design evolves, changing with it. The appropriate use of annotated UML diagrams can save time, which is a significant business benefit.

In addition, it becomes a pleasure to come back and add additional functionality at a later date.

Getting Started with the UML

Here's what worked for me and what I generally suggest to anyone interested in getting started:

- Get a tool such as MagicDraw UML (free community edition and trial available). Tools have a lot of intelligence built in, which helps you get up to speed quickly on a couple of diagram types. The tool knows the specification. Have a look at the various symbols available in one or two of the main diagram types described above. Start using some of the symbols; you don't have to use them all. Grow into the tool over time. Learn the structure of the documentation and start reading it.
- Get Martin Fowler's book, *UML Distilled*. Read it a couple of times. It's a great book, aptly titled. I found Martin's real-world experience and balanced view of using the UML very helpful.

Remember that the UML is a language, not a methodology, so don't think you have to change everything you already do successfully in order to get started with the UML. Take it one diagram at a time. Used effectively, the UML offers significant benefits.

Resources

- *MagicDraw UML (includes comprehensive documentation and examples):* www.magicdraw.com
- Fowler, M. (1999). *UML Distilled*. Addison-Wesley: www.pearsoned.co.uk/Bookshop/
- Sommerville, I. (2004). Software Engineering 7. Addison Wesley: www.pearsoned.co.uk/Bookshop/
- *UML 2.0 Specification and useful links:* www.uml.org

Attend the Second Annual

Register by September 9 to take advantage of the Super Early Bird Rate!

Software Test & Performance CONFERENCE

More than 60 keynotes, tutorials and classes!

SEE WHAT ATTENDEES SAID ABOUT LAST YEAR'S CONFERENCE:

"This was the most practical conference I have been to in 18 years."

Mary Schafrik, B2B Manager/QA & Defect Management
Fifth Third Bank

"Good for hands-on test managers and testers."

Steve Boykin, Deputy Program Manager
Citizenship and Immigration Services
Department of Homeland Security

"Extremely informative, and talking to colleagues in the field was incredibly enlightening."

Shari Pagley, Testing Supervisor
Sunrise Senior Living

"Definitely go for it. It's worth every minute and second of your time."

Felix Choy, Software Engineer
Avaya Inc.

"New tips and best practices in software testing were presented by excellent speakers and professionals."

Ella Naydorf, QA/Test Manager
Booz Allen Hamilton

"Very diverse set of sessions, passing on great and door-opening information and techniques at a level that many can comprehend. The speakers were excellent and approachable. The conference was very well organized!"

Jessica Navarette
Software Engineer
Alion Science & Technology

November 1-3, 2005
at the Roosevelt Hotel
New York City

www.stpcon.com • www.stpcon.com • www.stpcon.com • www.stpcon.com

Produced by

BZ Media SD Times Software Test & Performance

For information on exhibiting at or sponsoring the STP Conference, contact Donna Esposito at 415-785-3419 or desposito@bzmedia.com

Reflections on Debugging

Matching problem patterns to past issues

by Sachin Hejip

This rather pedagogically worded article is a collection of my thoughts on debugging Java software, the programming patterns I have used, some useful APIs, and techniques.

What it is not – it's definitely not complete in terms of information on debugging, its techniques, styles, etc. It's primarily a list of things that have worked for me time and again and a few tools that I keep in my toolkit to use when the situation demands it. I think they will be of use to you as well.

I have been fortunate to work in environments where I touched upon various facets of Java, used various APIs, and generally did extremely satisfying work. In all these years, debugging has stood out as an activity that everybody has to perform almost as much as they code or design. I have noticed time and again that being able to debug well is an extremely useful skill. It can be learned over time and honed and it is, to a large extent, the ability to match problem patterns to past issues. People like Rajiv (www.me.umn.edu/%7Eshivane/blogs/cafefeed/) can uncannily pinpoint a problem's cause when they hear its description. This ability comes from years of experience and the intent to learn from every new debugging experience.

Debugging

Debugging is the act of locating and fixing a flaw in software. A flaw can manifest itself in multiple ways. Sometimes it's apparent, for example, when the program crashes or does not do the intended action or does not return the intended result. Sometimes it is hard to say what's wrong when a program does not return, the CPU keeps processing something, or when the program does something unexpected in addition to the right action. Debugging, of course, is the action we take post having seen a flaw.

Isolating the Problem to Code: Identifying Where to Look for a Problem

The problem or flaw appears as a failure of the software to do something it should have. When you encounter a flaw, to debug it you need to form a mental model of the code to identify where the code is that failed. Debugging largely follows the process of elimination and this process is helped by any symptoms that you can find.

When you have the piece of code that failed, you try and find the cause by asking and answering questions – what is occurring? What possible causes could result in this problem? For example, if something should have happened and it didn't, perhaps the code was not reached. Why would the code not be reached? Maybe the *if* condition under which the method gets called did not evaluate to true or perhaps the *if (something != null)* check was called when *something* had the value *null*.

Another example: if there is an exception, then there is additional information about the location of failure and the steps that led to it. The type of exception will tell you the nature of failure. So if you see a *ConcurrentModificationException* thrown by an Iterator's next() method, you will have to:

1. Find out under what conditions this happens.
2. How could these conditions have been created in your program?
3. Maybe you removed something using *list.remove()* in your loop, or perhaps you passed reference to the list to some other thread that is modifying it.

Once you have a mental picture of the surroundings of the problem and why it might be occurring, it's a matter of eliminating the reasons one by one starting from the most likely cause.

Even if you intend to use a debugger, this is a necessary step. You have to always backtrack mentally from the point of failure to locate all possible causes of failure. A lot of debugging skill relies on this one ability alone.

Reading an Exception

Java Exceptions have a lot of information in them and should be well understood to debug problems. Often, I've noticed programmers use the following template for exceptions.

```
try {
// do stuff here
```

"I have noticed time and again that being able to debug well is an extremely useful skill"

Sachin Hejip, an architect with Sonic Software, is currently part of Sonic's ESB tooling initiative where he is leading a team of engineers to develop Eclipse plug-ins to take Sonic ESB development to the next level. A recipient of Pramati's highest award for technical excellence, the Pramati Fellowship, he has been a core member of the Pramati engineering team where he led the Web Server intiative and has been a key member of the Pramati Studio R&D team.

sachin.hejip@gmail.com

```
} catch(Exception ex) {
System.out.println(ex);
}
```

If you wish to print the exception to know when a problem has occurred, you must consider using *ex.printStackTrace()*. There are multiple advantages:

- When using System.out.println(ex), several times no message is printed other than the class name of the exception that occurred. Imagine having this piece of code in multiple locations; how will you ever know which catch handler printed *java.lang.NullPointerException*?
- An exception when printed stands out in a log file or the console. It's several lines long and just the pattern of an exception stack trace print is so different from the other message; it's much easier to find than an exception message that looks like other logging statements.
- Following JDK1.4, chained exceptions get printed as well and you don't have to manage them manually. Root cause gets carried along with the exception.
- Last and most important, the stack trace contains a wealth of information that can be used to create a mental picture of what happened.

There have been lots of times when I have looked at a stack trace and said: it should not have come here and been able to trace the problem to a wrong check in an earlier part of the code. You must know how to read an exception. Here's a Java exception printed out.

```
: Output generated by System.out.println()
:
java.lang.ArrayIndexOutOfBoundsException: 0

: Output generated by ex.printStackTrace()
:
java.lang.ArrayIndexOutOfBoundsException: 0
at com.sonicsw.tools.test.ThrowException.
processArgs(ThrowException.java:32)
at com.sonicsw.tools.test.ThrowException.
main(ThrowException.java:21)
```

1. ***java.lang.ArrayIndexOutOfBounds-Exception: 0***

The first part of printStackTrace is to do a print out of the exception similar to the System.out.println, so already you've gotten that for free. This part of the exception stack trace is formatted according to the type of exception, and the information printed varies from exception to exception. Some exceptions print nothing more than the class of the exception. Some exceptions (specially custom ones) print a lot of context information that led to this exception.

2. ***at com.sonicsw.tools.test.ThrowException.processArgs(ThrowException.java:32)***

The rest of the exception is the stack trace starting with the location that threw the exception at the top, and the caller of the method in which the exception was thrown below it, and so on until the executing thread's run method or the main method. The information provided

What is the easiest way to create mission-critical mobile messaging applications in Java and J2EE?

By using NCL's Provato™ !

Used by Operators, Aggregators, Content Providers and Wireless Application Developers to create and deploy mobile applications, **Provato™** gets the message across.

Provato connects with the message centers (SMSCs and MMSCs) of operators and aggregators over industry standard protocols. Developed in 100% java, it handles routing, concurrent applications, multiple networks, message persistence, failover, connection management and integrates with network management. With support for SMS, long messaging, Unicode, binary, WAP Push and Multimedia Messaging, NCL's Provato is endorsed by operators and aggregators worldwide. Provato runs as a standalone server or deploys on a J2EE application server. Application interfaces include JMS, SOAP/XML and HTTP. A user-friendly browser-based interface simplifies (remote) management and configuration - your java application need only call *send* and *receive*. So now you can focus on the business logic of your messaging application.

For more details visit **www.nclt.com/jdj**
NCL Technologies Limited
Tel: **+353 1 6761144** Email: **jdj@nclt.com**

ncl TECHNOLOGIES

on this line consists of:

- ***The fully qualified class name:*** com.sonicsw.tools.test.ThrowException
- ***The method:*** processArgs
- ***The file:*** ThrowException.java
- ***Line number:*** 32

Obviously, these are great nuggets of information. In some cases, when compilation does not include debugging information, you can end up with stack traces that don't have line numbers. That's usually a bummer but at least you have the stack of methods to locate where the problem occurred.

Other variations for method names are <clinit> for a static initializer – (this is also an example of exception chaining - note the "Caused by:"):

```
java.lang.ExceptionInInitializerError
Caused by: java.lang.
IllegalArgumentException
at com.sonicsw.tools.test.ThrowException.<c
linit>(ThrowException.java:21)
```

<init> for constructors and initializers:

```
java.lang.IllegalArgumentException
at com.sonicsw.tools.test.ThrowException.<i
nit>(ThrowException.java:20)
at com.sonicsw.tools.test.ThrowException.
main(ThrowException.java:24)
```

and the $number convention for anonymous classes:

```
java.lang.IllegalArgumentException
at com.sonicsw.tools.test.ThrowException$1.
actionPerformed(ThrowException.java:25)
at com.sonicsw.tools.test.ThrowException.(T
hrowException.java:23)
at com.sonicsw.tools.test.ThrowException.
main(ThrowException.java:31)
```

Of course, if the line number is available, it makes it a lot easier.

Sometimes, to understand why an exception occurred, you have to understand how control got to that point. The easy exceptions are always the ones that have a localized problem and you can easily catch the problem by the exception. But things can get really hard, for example, the culprit could lie several methods below and might not even be caused in this thread of execution. A previous event might have generated a bad value that was stored in a field and was picked up by this thread of execution in which the exception occurred. To get to the bottom of these, you need to create a mental back map of methods and events that might have occurred. It always helps to see what methods were called to get to the point of failure.

Trapping Exceptions

Sometimes, something you expected to execute does not occur and there is no information on why it failed. For some reason, control got transferred out of your method. Once you have eliminated any if conditions that are failing, it could be an exception getting thrown from somewhere deep inside your code. No exception gets printed though. This usually happens when you're implementing a piece that fits into a framework. Perhaps the framework has a logging switch that's set too low for exceptions to get printed, or the framework is faulty and is not printing exceptions being thrown by overridden methods. Maybe the framework is failing because of the unexpected exception being thrown by your method. In such cases, it's best to eliminate this possibility by wrapping the entire method by a try-catch block. For such debugging situations, I prefer to wrap with a *try … catch(Throwable t)* block because you want to be sure that no exception or error is being thrown. Sometimes, when errors get thrown (such as a *NoClassDefFoundError* because of a faulty classpath) it will slip through all *catch(Exception)* blocks.

Debugging is often an exercise in eliminating possibilities and locating the faulty piece of code. This is one technique to achieve that.

When to Use Thread.dumpStack

Another neat tool to keep in your toolkit is using *Thread.dumpStack()* or equivalently *new Exception().printStackTrace()*. Either of these methods do a *printStackTrace()* at this line without actually throwing an exception. The usual reason you do this is because you want to know what caused control of execution to come to this point. This can identify problems caused by a method being called unexpectedly.

Using VM Thread Dumps and Understanding Them

My biggest complaint with a lot of experienced Java developers is that they have never heard of the most amazing debugging tool called the VM Thread dump. You can use this technique in innovative ways:

- To detect deadlocks
- To diagnose UI hanging problems
- To diagnose slow UI issues
- To diagnose spinning/infinite loops
- For quick and dirty profiling
- To get an understanding of what the VM is doing at that instant

I can't do better than this excellent article on this topic available at www.me.umn.edu/%7Eshivane/blogs/cafefeed/2004/06/of-thread-dumps-and-stack-traces.html.

Classpath Problems

Another class of problems are classpath issues. There are times when you are not sure if there is another version of a class in the classpath that is getting picked up before yours, usually a result of a bad environment setup. To eliminate this possibility, a simple check is to add a print statement to see if your new code gets picked up. If it isn't getting picked up, you need to locate the other class that is getting picked up. One neat API in Java that allows you to locate where a class is being picked up is:

```
Class.getProtectionDomain().getCodeSource().
getLocation()
```

"When you encounter a flaw, to debug it you need to form a mental model of the code to identify where the code is that failed"

In most cases, depending on the class loader being used, you'll get the location of the class that is being executed and you can correct your environment setup.

When to Use a Debugger and When to Print

There are times when you should use a debugger and there are times when a print statement is more useful. You can use what is very well described by the brilliant pieces of work mentioned in the references section. The point I want to make here is that you must know what can be achieved using a debugger. It's an extremely powerful tool and can reduce debugging time quite a bit. It's applicable in many cases but is not suitable for a certain set of problems, such as threading issues or issues that show up in long-running tasks, in which case a log with good print statements is essential.

Good Debug Printing

Once you have a mental checklist of what the causes might be, you need to eliminate them. Debug prints can tell you quickly if your assumption is right or wrong. Ensure that your debug prints are not causing any side effects inadvertantly, for example:

```
System.out.println("value.getCode() = " + value.getCode());
if (value == null)
return;
switch(value.getCode()) {
…
}
```

It's important to print sufficient information about the object you are interested in. Perhaps your code is falling through a switch statement without firing any of the case clauses. You'll need to print the value of the switch condition. Think a little before you decide to type in any debug print statements. Often, when using good debug print statements, the code gets peppered at useful locations with debug statements that can be switched off with a boolean for future use. An easy to use pattern is:

```
private static boolean DEBUG = Boolean.getBoolean("<classname>.
debug").booleanValue();
```

If the class is com.test.ArgumentsProcessor, you would write

```
private static boolean DEBUG = Boolean.getBoolean("com.test.
ArgumentsProcessor.debug").booleanValue();
```

The advantage is that you can switch on debugging for this class without recompiling anything by specifying

```
Dcom.test.ArgumentsProcessor.debug=true when starting your
VM, e.g., java -Dcom.test.ArgumentsProcessor.debug=true <main-
class> <args>.
```

However, the compiler will not remove your debug statements during code optimization in this case.

Using Logging

There is not much that I can add on using logging that is not already covered by a vast amount of material. You can take a look at the References section below for other information on this subject. The point I would like to make here is that when you are writing print statements that go to a log for debugging purposes, maybe as a patch to a customer to diagnose a particularly tricky problem, think about how the log file might look when it's sent to you. What we think the output would be like when we write log print statements changes dramatically in a live system with multiple threads executing the same log messages. Maybe you need to print the thread ID to bunch all logs or an operation together, or perhaps some business data structure ID needs to be printed with each statement to understand what's happening. You may need to print out the execution path leading to the suspected problem location to know what conditions caused control flow to get there. Another thing to keep in mind is that log files can get so verbose and have so many messages that it becomes very difficult to scan them for problems later.

References

- Kernighan, B., and Pike, R. (1999). *The Art of Programming*. Addison-Wesley Professional. Highly recommended reading.
- Read, R.L. "How to Be a Program-mer." The first chapter is on debugging and is really well written: http://samizdat.mines.edu/howto/HowToBeAProgrammer.pdf

A Strategy for Aspect-Oriented Error Handling

Bringing new challenges to development

by Dan Stieglitz

Aspect-Oriented Programming (AOP) is a new, thought-provoking architecture paradigm still in its youth. One of AOP's primary goals is to improve the development of object-oriented systems by refactoring related lines of code that are typically found spread among classes (and are therefore difficult to maintain).

These blocks of related code represent functional "aspects" of the system, which now can be written in a single place and then "woven" into the target application. Logging the start and end of all method calls, securing method calls, and handling thrown exceptions are all commonly found aspects. While AOP provides an interesting and effective methodology for refactoring aspects out of code, how to implement these aspects is still left up to the developer. This article discusses a strategy for building a more easily maintainable, compartmentalized exception handing subsystem using a declarative chain of responsibility pattern and some concepts from aspect-oriented development. Many of the concepts discussed here have been implemented in an open source project called "Prob-lo-Matic" (http://problomatic.sourceforge.net).

As a consultant working on a wide spectrum of projects for various companies, I've found that subsystems specifically designed for exception handling (an aspect found in every project) can be strangely rare in the business world. There are few if any vendor products available that are "generic, full-featured exception handling suites." When exception-handling frameworks are home grown, as they often are, they usually contain a lot of difficult logic (causing more problems than they fix). There seems to be a wide variety of monitoring software that act as excellent exception detection and alerting tools but, in practice, few frameworks that provide application developers with structured, robust error-correction tools for use *within* their software packages.

For simple error handling, such as checking some variables or attempting to access an I/O device, the Java try/catch/finally system works fine. When an error occurs, there is usually no alternative but to handle the low-level exception inline and return some status to the calling stack frame. However, in modern distributed systems, there are more intricate requirements for error handling. Robust applications are required to retry failed steps, notify components of failures, and produce logging statements. First-rate application design demands a generic, pluggable exception-handling framework for our distributed systems. A declarative chain of responsibility pattern can act as the foundation for building such a system, and AOP can give us a powerful non-invasive way to integrate such a system with our code.

Dan Stieglitz is an independent software consultant in New York. He specializes in designing and developing distributed applications in Java and J2EE.

dan@stieglitech.com

Chain of Responsibility (a.k.a. Chain of Command)

The chain of responsibility pattern's intent is to "avoid coupling the sender of a request to its receiver by giving more than one object a chance to handle the request" (Gamma, et al. *Design Patterns*). Essentially, a message object is passed through a chain of objects implementing a common interface. The interface includes methods to chain the handler together, i.e., get- and setSuccessor methods:

```
public interface Handler {
public void handle(Object message);
public void setSuccessor(Handler successor);
public Handler getSuccessor();
}
```

Implementations of this interface use the successor property to pass the message through the chain, as in:

```
public class HandlerImplementation {
public void handle(Object message) {

// local handling code here

if (getSuccessor()!=null) {
successor.handle(message);
}
}
}
```

Handlers are acquired by the application in some fashion (more on that later), and execution is branched to the handle's handle() method. In the case of an exception-handling framework, we wish to construct an object that encapsulates all of the information about the state of our application when an error occurred, as well as information about what to do for specific errors. For example, if we have implemented a handler to retry database calls it's necessary to provide information as to where the secondary database instances are. To do so, it's useful to create an object tree based on a message interface that's understandable by all handlers.

The Problem Interface

An important part of using the chain of responsibility pattern is standardizing the interface of the message passed down the links. If a wrapper around the underlying exception, error or application-generated warning is applied, information can be shared among nodes. An example interface could be:

```
public interface Problem {
public void setAttribute(String name, Object obj);
public Object getAttribute[(String name);
public boolean hasAttribute(String name);
}
```

Implementations of Problem could be created for specific recurring situations, such as:

```
public class DatabaseAccessProblem implements Problem { … }
public class ProblemInRequestProcessor implements Problem { … }
public class SecurityProblem implements Problem { … }
```

Each specific implementation can store information that allows for some logic to occur. Coupled with a declarative handler chain (see below), a framework evolves that allows for robust, extensible generic exception processing subsystems to be designed and quickly adapted to any project.

Declarative Programming and Prob-lo-Matic

Declarative programming refers to the practice of refactoring application logic out of the code and into a reference file. An excellent example of a declarative framework is the Spring Framework (http://www.springframework.org/), which allows developers to move dependencies out of code and into XML files (the pattern on which the framework is based is called "dependency injection" or "inversion of control"). A declarative approach facilitates quick system maintenance and the ability to "hot swap" features of a deployed application.

Using the inversion of control pattern makes code very flexible and is the method used by Prob-lo-Matic to configure the problem handler chains. Any number of formats could be used to store this information; for example, here is a Prob-lo-Matic configuration file, which is XML (see Listing 1).

The configuration and instantiation of problem handler chains is where Prob-lo-Matic adds value. Prob-lo-Matic basically provides a factory for creating chains-of-command based on Problem classes. It exposes a static method void handleProblem(Problem aProblem) that constructs a chain based on the XML configuration file. This method then passes the specified Problem to the first link in the chain for processing. To modify the behavior of the exception handling in your application, you need only modify the Prob-lo-Matic configuration file and the changes will be reflected in your application.

To use Prob-lo-Matic you could refer to the Prob-lo-Matic classes in your application. While this is a fine solution for new or simple applications, we might want to be able to apply this framework to preexisting code and to do so in a way that doesn't disturb the integrity of the existing application. This would have been a difficult or impossible task in the past, but AOP gives us some interesting options for a clean integration of this framework.

Two AOP Approaches: Spring AOP and Bytecode Instrumentation

If you're new to AOP it may take a while to wrap your head around the concepts, especially when you see that the AOP designers have developed a whole new lexicon of AOP terms to describe the new concepts. This article doesn't purport to explain even a fraction of AOP, but we will glance over a few AOP concepts and discuss the Spring framework's implementation of a subset of AOP. Spring's implementation can be used to "weave" our error handling code into our application. We are interested in intercepting thrown exceptions, wrapping them in our Problem object, and passing them off to handlers. To this end, we can implement what's called a *throws advice* in Spring. This is implemented using Prob-lo-Matic as in the following example:

```
public class ExceptionInterceptor implements ThrowsAdvice {
 public void afterThrowing(Method m, Object[] args, Object target,
Exception ex) {
   Problem problem = new RawProblem(ex);
   problem.setAttribute("method.name",m.getName());
   // set other attributes here
   Problomatic.handleProblem(problem);
 }
}
```

The drawback with this approach is that our target object must be a JavaBean and also configured using the Spring Framework. If this is the case, we can tell Spring to intercept all thrown exceptions and call the afterThrowing method. For more information on integrating throws advice with Spring-enabled beans, see the Spring Framework documentation on http://www.springframework.org/.

Alternatively, we can weave this code into our application using another technique called *bytecode instrumentation*. This involves inserting lines of code into Java classes with a tool after the javac compiler has compiled them. Prob-lo-Matic provides such a tool called Problomatic-Weaver and a corresponding Ant task, WeaveTask. The ProblomaticWeaver searches for all try/catch blocks in the specified classes and inserts a call to Prob-lo-Matic before any other code in the catch block (any code already present in the catch block is preserved after this call). For example, the following code fragment is written:

```
public class MyClass {
public void myMethod() {
Try {
    do();
} catch (Throwable t) {
    System.out.println(t.getMessage());
}
}
```

After compilation, the Prob-lo-Matic code is woven into MyClass.class. If this class was decompiled, it would look like this:

```
public class MyClass {
public void myMethod() {
Try {
    do();
} catch (Throwable t) {
 RawProblem problem = new RawProblem(t);
 Problomatic.handleProblem(problem);
    System.out.println(t.getMessage());
}
}
}
```

Practical Example: Retrying Database Calls

Once Prob-lo-Matic has been integrated into our code, we can use it to perform a number of useful functions when exceptions are thrown. A common requirement for reliable database access is to retry connecting to a database when a connection fails, or to switch to an alternate database if the desired database is down for some reason. Consider a set of implementations of Problem that contain context-specific callback methods on their source, as defined in an interface such as:

```
public interface Recoverable {
 public int getMaximumRecoveryAttempts();
 public int getRecoveryAttemptsCount();
 public void attemptedRecovery();
 public boolean canRecover();
 public void setConnection(Connection con);
 public void setConnected(boolean yesOrNo);
}
```

An object that implements Recoverable (see Listing 2) encapsulates the data and logic required to manage multiple attempts of some procedure. If our business objects (or DAOs) implement the Recoverable interface, we can define a DatabaseProblem that has a reference to our Recoverable instance. Our ProblemHandler implementation (see Listing 3) then uses these callback methods to instruct the Recoverable object how to recover from the problem. In addition, the state of the recovery (number of times tried, etc.) is managed by the DatabaseProblemHandler. Our Prob-lo-Matic configuration looks like this:

```
<problomatic-configuration>
     <define-chain problem="com.mypackage.DatabaseProblem">
           <chain-link
handler="com.mypackage.DatabaseRetryHandler"
           <chain-link
handler="com.stieglitech.problomatic.handlers.EmailNotificationHan
dler"/>
</define-chain>
</problomatic-configuration>
```

When the execute() method of DatabaseFailover is called it will attempt to create a connection getMaximumRecoveryAttempts() or until isConnected(), whichever comes first. When an exception is thrown while trying to get a connection, Prob-lo-Matic is invoked. The DatabaseRetryHandler will be loaded with the DatabaseProblem passed in, and callbacks will be made to DatabaseFailover with new connections that are made by the handler. This approach has given us clean encapsulation of our error-handling logic, and the ability to easily maintain or augment this logic. Using AOP, we can weave this functionality into our system with little impact on the core system code.

Summary

AOP provides developers with a new and exciting methodology that builds on top of OOP concepts, and tries to mitigate some of the inherent problems that OOP brings to large, complex projects. AOP will also bring new challenges to development, especially when it comes to implementing aspects in effective and maintainable ways. Error handling is clearly an important aspect for many applications, especially those with high reliability requirements. An extensible framework for error handling such as Prob-lo-Matic, integrated into applications with AOP, can add a good deal of value to any development effort. Prob-lo-Matic, an open source work in progress, can be downloaded from http://problomatic.sourceforge.net/.

Listing 1

```
<problomatic-configuration>
<default-properties
<handler="com.stieglitech.problomatic.handlers.EmailNot
ificationHandler">
                    <property name="mail.smtp.host"
value="my.mail.server"/>
      </handler>
</default-properties>
     <define-chain problem="com.stieglitech.problomat-
ic.problems.RawProblem">
          <chain-link
handler="com.stieglitech.problomatic.handlers.
SystemPrintlnHandler"/>
          <chain-link
handler="com.stiegltiech.problematic.handlers.EmailNoti
ficationHandler"/>
</define-chain>
</problomatic-configuration>
```

Listing 2

```
public class DatabaseFailover implements Recoverable {
 private int recoveryAttempts = 0;
 private java.sql.Connection con;
 private boolean isConnected = false;

 public void attemptedRecovery() {
  recoveryAttempts++;
 }

 public boolean canRecover() {
  if (getRecoveryAttemptsCount() < getMaximumRecovery-
Attempts()) {
    return true;
  } else {
   return false;
  }
 }

 public void doDatabaseWork() { // insert, delete, update,
whatever }

 public void connectToDatabase() {
  try {
   con = DriverManager.getConnection(PRIMARY_DB_URL, DB_
USER,
     DB_PASS));
  } catch (SQLException e) {
   DatabaseProblem prob = new DatabaseProblem(e);
   prob.setSource(this);
   Problomatic.handleProblem(prob);
  }
 }

 public void execute() {
  while (canRecover() && !isConnected()) {
   connectToDatabase();
  }
  doDatabaseWork(getConnection());
  closeConnection();
 }

 public void setConnection(Connection con) {
  this.connection = con;
 }
}
```

Listing 3

```
public class DatabaseRetryHandler
implements com.stieglitech.problomatic.ProblemHandler {

 public void handleProblem(Problem aProblem) {
  if (aProblem instanceof DatabaseProblem) {
   DatabaseProblem dbProb = (DatabaseProblem) aProblem;
   Recoverable source = (Recoverable) dbProb.getSource();
   Connection con = getAlternateConnection();
   if (con!=null) {
    source.setConnection(con);
    source.setConnected(true);
}
else {
 source.setConnected(false);
}
source.attemptedRecovery();
  }
 }

 private Connection getAlternateConnection() { // get alter-
nate connection }
}
```

*Dynamic***PDF**™ Merger v3.0
Our flexible and highly efficient class library for manipulating and adding new content to existing PDFs is available natively for Java

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

*Dynamic***PDF**™ Generator v3.0
Our popular and highly efficient class library for real time PDF creation is available natively for Java

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

Try our free Community Edition!

*Dynamic***PDF**
www.dynamicpdf.com

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

ceTesoftware
INFINITE POSSIBILITIES
800.631.5006 +1 301.656.1331

The Simplicity of **EJB 3.0**

A step in the right direction

by Raghu Kodali

Over the past few years, the Enterprise JavaBeans (EJB) specification has evolved significantly. In the early days of EJB, application developers faced a burden of overwhelming complexity: they had to manage several component interfaces, deployment descriptors, and unnecessary callback methods; work within the limitations of the EJB Query Language (EJBQL); and learn and implement the design patterns used to overcome the limitations of the specification.

The introduction of the EJB 2.1 specification did improve things, although many still say the specification is too complex – and that criticism is often seen as a reflection of the problems of the entire J2EE platform.

The next major release of the J2EE 5 platform is focused on ease of development. As a cornerstone of the platform, much of the effort centers on reducing the complexity of EJB. The EJB 3.0 specification simplifies development by removing the requirements for interfaces, deployment descriptors, and callback methods and by adopting regular Java classes and business interfaces as EJBs.

The specification also leverages metadata annotations that are standardized with JSR-175, and the proven Plain Old Java Object (POJO) persistence architecture used by object-relational (O/R) frameworks such as Oracle TopLink and Hibernate. These last two features have greatly reduced much of the specification's complexity. Now you can take a regular Java class, add annotations to it, and deploy it to an EJB 3.0 container as an entity. A configuration by exception approach is taken so that the container accepts the defaults whenever possible.

```
Sample Entity Bean with Annotations

@Entity
@Table(name="PLAYER", schema="CMPROSTER")
@NamedQuery(name="findAll",queryString="SELE
CT OBJECT(p) FROM Player p");
public class Player implements Serializable
{
 @Id
  @Column(name = "ID", primaryKey = true,
nullable = false)
  public String getId()
  {
    return id;
  }
//………………..
}
```

The features I've mentioned above are only the tip of the iceberg – the EJB 3.0 specification provides a slew of new features and enhancements.

This all sounds great on paper, but I wanted to find out just how much easier it is to develop applications with EJB 3.0. So, I decided to give the specification a spin and see for myself. I chose an existing EJB 2.1 application that implements some common use cases with design patterns such as a Session façade, and migrated the application using the new features of EJB 3.0. I used the publicly available demo application RosterApp (included with the J2EE 1.4 tutorials), which lets you maintain team rosters for players in leagues.

I took the bottom-up approach to migrate RosterApp with EJB 3.0 technology, starting with:

- Entity beans
- Data transfer objects (DTO)
- Session bean
- Utility and client classes

Migrating the Entity Beans

RosterApp has three entity beans: LeagueBean, TeamBean, and PlayerBean. Instead of taking the existing beans, deleting the home and local interfaces, and converting the abstract methods to getter and setter methods with annotations, I reverse-engineered the RosterApp tables from an Oracle Database 10*g* as EJB 3.0 entities. My result was three simple POJOs (League, Player, and Team) with a set of default annotations. All I had to do was add annotations for the many-to-many relationship between Player and Team. The annotations look like Listing 1.

The EJB 3.0 specification lets you specify O-R metadata via annotations. It provides a wide range of annotations that cover different types of relationships between POJOs, constraints, column information, sequence generators, composite primary key, and inheritance.

Once you migrate all the O/R mappings as annotations in the POJOs, the next step is to convert a bunch of finder methods with EJBQL from EJB 2.1 to new POJOs. Most of these finder methods were already defined for the player bean. EJB 3.0 provides the NamedQueries annotation to group together individual NamedQuery objects. I took all the EJBQL from the existing application and created a NamedQueries annotation, which looks like Listing 2.

The EJB 3.0 specification provides a Query API that can be used for both static and dynamic queries. A named query can be defined as a standalone query or attached to a query method of the bean class. You can define named queries in EJBQL or SQL. This is a boon for Java developers familiar with SQL syntax, as they can become EJB developers without having to learn another query language.

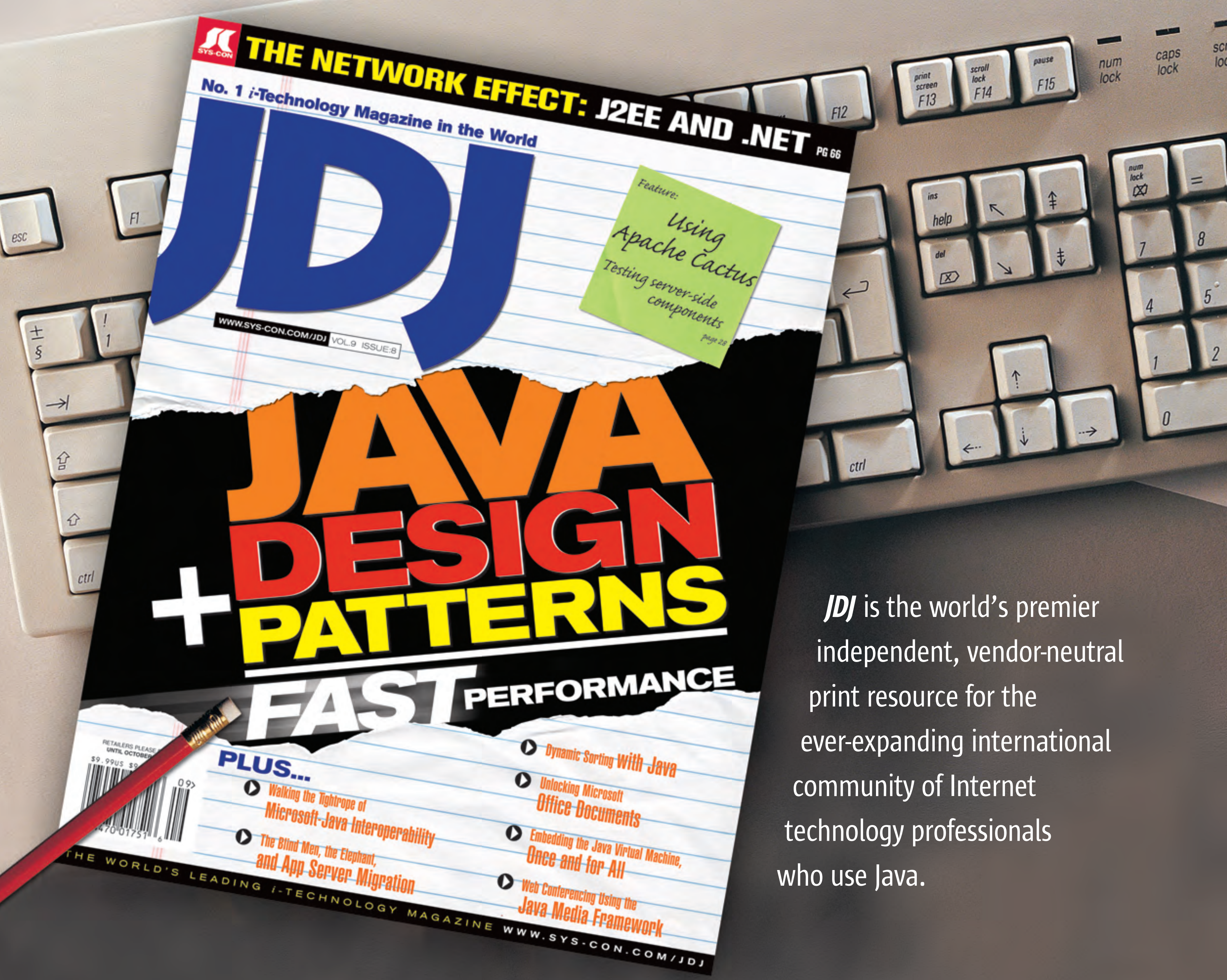
Mappings and finders covered almost 90–95% of the entity bean migration. The remaining part of the project consisted of ejbSelect statements and methods that perform add and remove operations on the Team POJO. I needed to simplify these methods. The following code shows one of the methods before and after migration. ejbSelect methods were migrated as NamedQuery in the Session facade (which is discussed later in this article).

```
// remove operation on Player before migration

public void dropPlayer(Player player)
{
```

Raghu R. Kodali is consulting product manager and SOA evangelist for Oracle Application Server. He leads next-generation SOA initiatives and J2EE feature sets for Oracle Application Server, with particular expertise in EJB, J2EE deployment, Web services, and BPEL. Prior to product management, Raghu held presales and technical marketing positions in Oracle Asia-Pacific, based in Singapore. Prior to Oracle, he worked as asoftware developer with National Computer Systems, Singapore. He holds a master's degree in computer science and is a frequent speaker at technology conferences. He maintains an active blog at Loosely Coupled Corner.

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


SYS-CON
THE NETWORK EFFECT: J2EE AND .NET PG 66
No. 1 i-Technology Magazine in the World
JDJ
WWW.SYS-CON.COM/JDJ VOL.9 ISSUE:8
Feature: Using Apache Cactus
Testing server-side components page 28
JAVA DESIGN + PATTERNS
FAST PERFORMANCE
PLUS...
Walking the Tightrope of Microsoft-Java Interoperability
The Blind Men, the Elephant, and App Server Migration
Dynamic Sorting with Java
Unlocking Microsoft Office Documents
Embedding the Java Virtual Machine, Once and for All
Web Conferencing Using the Java Media Framework
THE WORLD'S LEADING i-TECHNOLOGY MAGAZINE WWW.SYS-CON.COM/JDJ

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

Only $69^99 ONE YEAR 12 ISSUES

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

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

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

OFFER SUBJECT TO CHANGE WITHOUT NOTICE

```
Debug.print("TeamBean dropPlayer");
try {
Collection players = getPlayers();
players.remove(player);
}
catch (Exception ex) {
throw new EJBException(ex.getMessage());
}
}
//remove operation after migration

public void dropPlayer(Player player) {
Debug.print("TeamBean dropPlayer");
getPlayers().remove(player);
}
```

Migrating DTOs

DTOs are the next layer in RosterApp. The entities in EJB 3.0 are POJOs; you can directly transfer them between the business and client tiers without first having to create a separate set or layer of classes as in EJB 2.1. The existing RosterApp used DTOs to transfer Teams, Players, and Leagues data collections between the client and Session facade. The new EntityManager API in the EJB 3.0 persistence specification, which is used to create, remove, find, and query entities, works nicely to attach and detach objects from the persistence context. The EntityManager's merge operation lets you propagate state from detached entities onto persistent entities managed by the EntityManager.

The EJB 3.0 RosterApp didn't require the existing DTO baggage, but I had to make sure that the Team, League, and Player POJOs implemented java.io.Serializable. I also had to get rid of extra methods such as getPlayersofTeamCopy() in the Session facade of the EJB 2.1 RosterApp, which were doing the grunt work of managing data between DTOs and entity beans. On top of eliminating the extra overhead, I had to simplify the business methods in the Session façade of the EJB 3.0 RosterApp, as they were using DTOs all over the place. Listing 3 shows the sample migrated code.

Migrating the Session Beans

After eliminating the DTOs from RosterApp, I migrated the Session façade (RosterBean). First, I had to remove the home interface and clean up the remote interface to make it a business interface. To do so, I made sure the interface wasn't extending EJBObject and was annotated with @Remote. I annotated the bean class with @Stateless. The RosterBean in EJB 2.1 had a number of business methods that interacted with the Team, League, and Player entity beans. The largest chunk of the porting exercise was simplifying the business methods to use the EntityManager API, creating NamedQueries for ejbSelect methods, setting up the parameters for standalone named queries that were defined in POJOs, and making sure the methods didn't interact with DTOs that had already been removed (see Listing 4). (Listings 4–5 can be downloaded from www.jdj.sys-con.com.)

Migrating the Client

Once the Session façade task was completed, it was time to clean up the client code. The main difference between the EJB 2.1 and EJB 3.0 client is the lookup code, and minor changes to make sure the objects returned by the business methods were POJOs instead of DTOs (see Listing 5).

Comparing EJB 2.1 to EJB 3.0

Once the migration was done, I compared the lines of code and the number of Java and XML files in both versions. The biggest difference between the existing EJB 2.1 and new EJB 3.0 application was the number of descriptors. The EJB 2.1 application had a number of deployment descriptors, while the newer application had eliminated all of them except for application-client.xml and application.xml (see Table 1).

I used the numlines (www.gammadyne.com/cmdline.htm) utility, which gives the line count for uncommented and non-blank lines – the only types of lines added for the old and new applications. In the EJB 2.1 application, XML files were counted based on the deployment steps recommended in the J2EE 1.4 tutorial (see Table 2).

	EJB 2.1	EJB 3.0
Number of Java Files	17	7
Number of XML Files	9	2

Table 1

	EJB 2.1	EJB 3.0
Lines of Code	987 in 17 Java files	716 in 7 Java files
Lines of Code	792 in 9 XML files	26 in 2 XML files

Table 2

Conclusion

EJB 3.0 definitely makes it easier to develop entity and session beans, thanks to its simplified model and its leverage of well-known artifacts like POJOs and interfaces. The new EntityManager API is a plus; I was able to change the business methods quite easily without reading the specification.

There are other neat features in EJB 3.0, such as the ability to use database sequences; I used this for one of the POJOs, but backed out the changes to make the existing and new applications more similar. There doesn't seem to be support for native SQL queries, though the specification claims there is. I would have loved to use those queries instead of EJBQL, as database portability wasn't an issue for this exercise.

While the specification is a step in the right direction, I'd like to see more tool/IDE support for EJB 3.0 so that more Java application developers can get up to speed on it. While any standard IDE with decent support for Java SE 5.0 will be a good start, I'd like to see better tooling to support complex mappings (such as many-to-many), and to facilitate inline or immediate feedback on the validity of NamedQueries instead of waiting for deployment.

Maintaining applications can't be ignored, as applications will live for few years after the development cycle. All the features that make application development easier will also provide returns in the application maintenance cycle.

I recommend that developers take a fresh and unbiased look at the EJB 3.0 specification by checking out its features and giving the publicly available EJB 3.0 containers a spin.

(Oracle Application Server EJB 3.0 Preview was used for this exercise.)

References

- *Software used for EJB 2.1 application (as recommended in the J2EE 1.4 tutorial):* http://java.sun.com/j2ee/1.4/docs/tutorial/doc/About.html#wp80138
- *Software used for EJB 3.0 application (Oracle Application Server EJB 3.0 Preview):* www.oracle.com/technology/tech/java/ejb30.html
- *Source files for the migrated application:* www.jroller.com/resources/raghukodali/howtoRosterApp.zip
- *EJB 3.0 specification:* http://java.sun.com/products/ejb/docs.html

Listing 1

```
//Team POJO

@ManyToMany(cascade=PERSIST,fetch=EAGER)
@AssociationTable(table=@Table(name="TEAM_PLAYER"),
joinColumns=@JoinColumn(name="TEAM_ID", referencedColumnName="ID"),
inverseJoinColumns=@JoinColumn(name="PLAYER_ID", referencedColumnNam
e="ID"))

public List<Player> getPlayers() {
return players;
}
// Player POJO

// players is a List in the Team POJO

@ManyToMany(mappedBy="players", fetch=EAGER)

public List<Team> getTeams() {
return teams;
}
```

Listing 2

```
// NamedQueries in Player POJO

@NamedQueries
({
@NamedQuery(name="findAll",queryString="SELECT OBJECT(p) FROM Player
p"),
@NamedQuery(name="findByCity",queryString="SELECT DISTINCT OBJECT(p)
FROM Player p, in (p.teams) as t where t.city = :city"),
@NamedQuery(name="findByHigherSalary",queryString="SELECT DISTINCT
OBJECT(p1)FROM Player p1, Player p2 WHERE p1.salary > p2.salary AND
p2.name = :name "),
@NamedQuery(name="findByLeague", queryString=" select distinct
object(p) from Player p, in (p.teams) as t where t.league = :
league"),
@NamedQuery(name="findByPosition", queryString=" select distinct
object(p) from Player p where p.position = :position"),
@NamedQuery(name="findByPositionAndName",queryString=" select distinct
object(p) from Player p where p.position = :position and p.name = :
name"),
@NamedQuery(name="findBySalaryRange",queryString="select distinct
object(p) from Player p where p.salary between ?1 and ?2"),
@NamedQuery(name="findBySport", queryString="select distinct object(p)
from Player p, in (p.teams) as t where t.league.sport = ?1"),
@NamedQuery(name="findByTest", queryString=" select distinct object(p)
from Player p where p.name = ?1"),
@NamedQuery(name="findNotOnTeam",queryString=" select object(p) from
Player p where p.teams is empty")
})
```

Listing 3

```
//code before migrating to EJB 3.0

public List getTeamsOfLeague(String leagueId)
{
Debug.print("RosterBean getTeamsOfLeague");
ArrayList detailsList = new ArrayList();
Collection teams = null;
try {
LocalLeague league = leagueHome.findByPrimaryKey(leagueId);

teams = league.getTeams();
} catch (Exception ex) {
throw new EJBException(ex.getMessage());
}

Iterator i = teams.iterator();

while (i.hasNext()) {
LocalTeam team = (LocalTeam) i.next();
TeamDetails details =
new TeamDetails(team.getTeamId(), team.getName(), team.getCity());

detailsList.add(details);
}

return detailsList;
}
//code after migrating to EJB 3.0

 public List getTeamsOfLeague(String leagueId)
{
Debug.print("RosterBean getTeamsOfLeague");
League l = (League)getEntityManager().find("League", leagueId);
return l.getTeamList();
}
```

COMMON CONTROLS www.common-controls.com

The Java™ Presentation framework for J2EE™ Web applications

Based on:
- Java™
- Servlets™
- Java Serverpages™
- and Struts

For Free!

Get your free trial version – www.common-controls.com

See the common controls in action – go for the Online Demo!

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

Lists | Trees | Tabfolders | Menu | Forms | BreadCrumbs | Calendar | Colorpicker

www.common-controls.com

Joe Winchester
Desktop Java Editor

J2SE and Open Source – Living Together in Perfect Harmony

Open source and J2SE,
living together in perfect harmony
Side by side on my computer keyboard,
Oh yeah, why can't we?

Java has been the springboard for some of the most successful open source projects today including JBoss, NetBeans, and Eclipse. Several folks though have felt the missing piece was an actual open source implementation of the runtime. Some view Sun's stewardship of Java and the JCP as being too controlling, while others believe it is an essential benign rule that preserves the integrity of the language. The view taken by Sun execs is that the JCP is essential to preserve portability while preventing forks in the language (as Microsoft once attempted with proprietary extensions). Should these occur, they could compromise a user's expectation that a Java program, once written, can run anywhere. Advocates of open source, however, deny this would occur and cite successful projects in which the benefits of fast innovation and widespread adoption have been fueled by having an open source community of developers who drive the implementation forward.

Earlier this year the Apache Foundation announced the Harmony project. Its aims are to write an open source version of J2SE from scratch. Given that specifications for J2SE are freely available, one question might be why such a move hasn't occurred before? In the FAQ for Harmony, Gier Magnusson from Apache states: *"While the Java Community Process has allowed open source implementations of JSRs for a few years now, Java 5 is the first of the J2SE specs that we are able to do due to licensing reasons"* (http://mail-archives.apache.org/mod_mbox/incubator-general/200505.mbox/%3CE3603144-2C26-4C31-896D-6CC7445A63EB@apache.org%3E).

Rather than reinvent the wheel, after the initial announcement it seems that the plan is to try to pollinate the project with code and ideas from existing Java runtimes. At JavaOne Geir said: *"There is a lot of software out there that we are hoping can be donated. We are hoping that we will get seeded with some code from existing production Virtual Machines."* This is an invitation to the big guns of software and mom and pop developers, many of whom have either privately or openly tinkered with creating their own implementations of Java.

There are several potential Harmony contributors who might be in Geir's sights to step up to the plate. BEA has a very good JVM implementation in JRockit (http://www.bea.com/framework.jsp?CNT=index.htm&FP=/content/products/jrockit), while Kaffe (http://www.kaffe.org/) is a clean room implementation of the JVM.

Making Harmony succeed, however, requires more than just a JVM, as J2SE contains many lines of code in the actual class libraries, all of which need API compatible reimplementation. Due to licensing and IP issues this most likely must be done in an environment without reference to the existing Sun J2SE source code base, so the hill to climb is steep and high one.

I usually don't pay much attention to the cries of doom issued by execs of any company that has a vested interest in whether open source Java should succeed or fail, but I was surprised to hear James Gosling, when asked about Harmony, raise the issue of why Apache wants the Harmony license to be different from the existing J2SE one. He said: *"I understand why they would like it to be different. From our point of view that would actually be more destructive than helpful. It boils down to forking: they believe that the ability to fork is an absolutely critical right"* (http://www2.vnunet.com/lite/News/1163182). His voice is one I respect and should be listened to as words of wisdom and caution so the worst case scenario doesn't become a reality. To ensure compatibility with J2SE, however, the harmony FAQ states that it will be verified against the J2SE Test Compatibility Kit. The TCK is published by the JCP and is the yardstick used by all ratified J2SE integrations (JCP or commercial) that to date has ensured no fragmentation exists and no compromise of integrity has occurred.

Whatever the outcome of the Harmony project, while I do understand the concerns of those who are frightened of fragmentation in the language, I am more excited by the prospects of having an open source implementation J2SE. This will enjoy the same benefits that other open source projects have experienced, where a community of like-minded, smart individuals who share the same goal can participate equally in the same code and knowledge base of ideas and innovation. Ideas, bug reports, and schedules are transparent and can be iterated in public toward the best solution. The momentum of the project becomes fueled by the feedback loop created by an alliance of like-minded individuals in a partnership of conviction. To those who don't believe this will work, ask not if the glass is half empty, but listen instead to the lyrical genius of Sir Paul McCartney and Stevie Wonder:

We all know that people are the same
where ever you go
There is good and bad in everyone
Learn to live, we learn to give each other
What we need to survive, together alive

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

joewinchester@sys-con.com

FrameResizer

Resizing windows

by Phil Herold

This article presents a Java/Swing component implementation of a feature that is ubiquitous in nearly all desktop applications, particularly Windows applications – an area in the lower right portion of a window (Frame) that can be used to resize the window.

Of course, a window can be resized with most desktop managers by dragging the lower-right edge – the additional component simply serves as a visual indicator of the resize capability, and also increases the margin of error for the mouse drag.

Typically this component is placed in a status or message area at the bottom of a window. I demonstrate my solution in the context of a very simple (and barely functional) Web browser. I also introduce a technique in which a Frame/JFrame/JInternalFrame can exhibit "continuous layout" behavior as it is resized.

Details

The component is called FrameResizer. It subclasses JComponent and has two main jobs that it performs. First, it draws itself and, second, it handles mouse events so that it can both change the mouse cursor (on mouse enter/leave) and resize the window (on mouse press/drag/release). FrameResizer can be used to resize a Frame, JFrame, or JInternalFrame, each of which has a common ancestor of java.awt.Container in the component class hierarchy. So constructors for FrameResizer take a Container argument as follows:

```
public FrameResizer(Container parent);
public FrameResizer(Container parent, bool-
ean useContinuousLayout);
```

Rendering

Figure 1 shows the FrameResizer in its typical context, at the lower right edge of a window, as part of a status/message area.

FrameResizer draws itself in the paintComponent() method as shown in Listing 1. There are seven "ribs" that make up the component.

A rib is one of the raised "bumps" in the component visual; it's actually just two 2x2 pixel rectangles drawn in different colors offset slightly from each other. You can see this in the exploded view of the FrameResizer in Figure 2.

Resizing

Much of the interesting FrameResizer code is in the MouseInputListener inner class that the component adds to itself. In the mouseEntered() and mouseExited() methods, the cursor is changed to Cursor.NW_RESIZE_CURSOR and the default Cursor, respectively. In the mousePressed() method (see Listing 2), the mouse location is converted to actual screen coordinates, and the relative location from edge of the window (the Point mouseAdjust) is calculated.

In the mouseDragged() method (see Listing 3), the current mouse position is again converted to absolute screen position, and the new bounds of the parent Container (Frame/InternalFrame) are recalculated based on the original mouse location.

The mouseReleased() method calls mouseDragged() for a final calculation, and then "validates" the parent container. Since FrameResizer supports either an external Frame

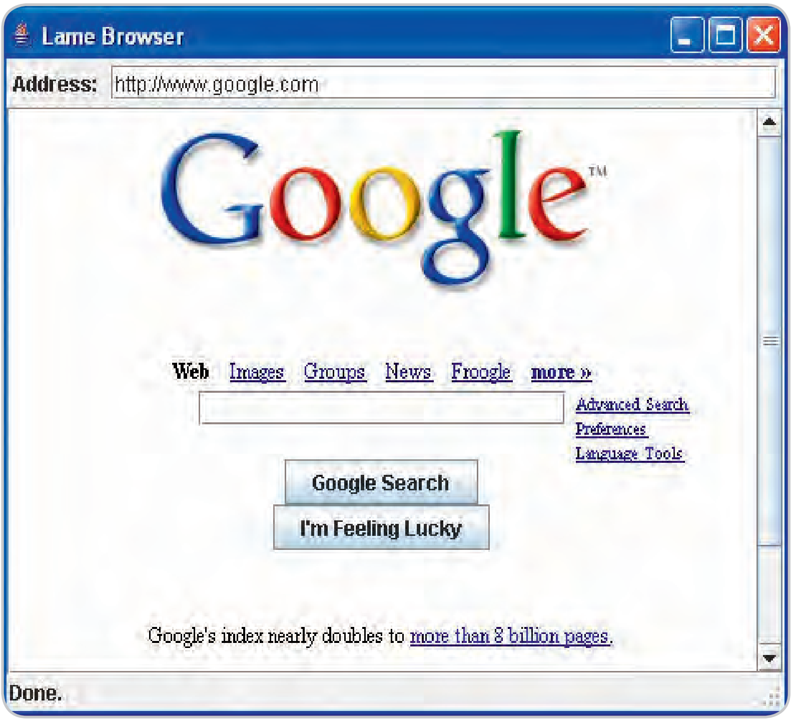

Lame Browser
Address: http://www.google.com
Google™
Web Images Groups News Froogle more »
Advanced Search
Preferences
Language Tools
Google Search
I'm Feeling Lucky
Google's index nearly doubles to more than 8 billion pages.
Done.

Figure 1 FrameResizer

Phil Herold is a Java architect with over 24 years experience in software engineering. He has been working with Java client technologies since 1996.

Phil.Herold@sas.com

or a JInternalFrame, the validate() method does the appropriate validation and invokes the repaint() method (see Listing 4). This needs to be done at the end of the AWT EventDispatchThread (thus the use of SwingUtilities.invokeLater()). This method is declared static so it can be re-used for the "continuous layout" functionality (see below).

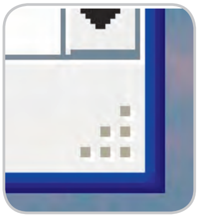


Figure 2 Exploded View

Continuous Layout

This concept is familiar to Swing developers who use JSplitPane – it's the ability to have internal components validate and repaint as the container is resized (in real time). Unfortunately, when a top-level Java desktop window is resized, this doesn't happen, as the validate/repaint messages are not sent to the underlying Container until resizing (via mouse dragging) is complete. Figure 3 is a capture of our browser window during the mouse drag. You can see how the contents of the window are not "stretched' to fit the bounds of the Frame – this is done when the mouse is released.

Ideally you want the contents of the window to stretch to the frame bounds during resizing. This capability is evident in most non-Java top-level desktop windows.

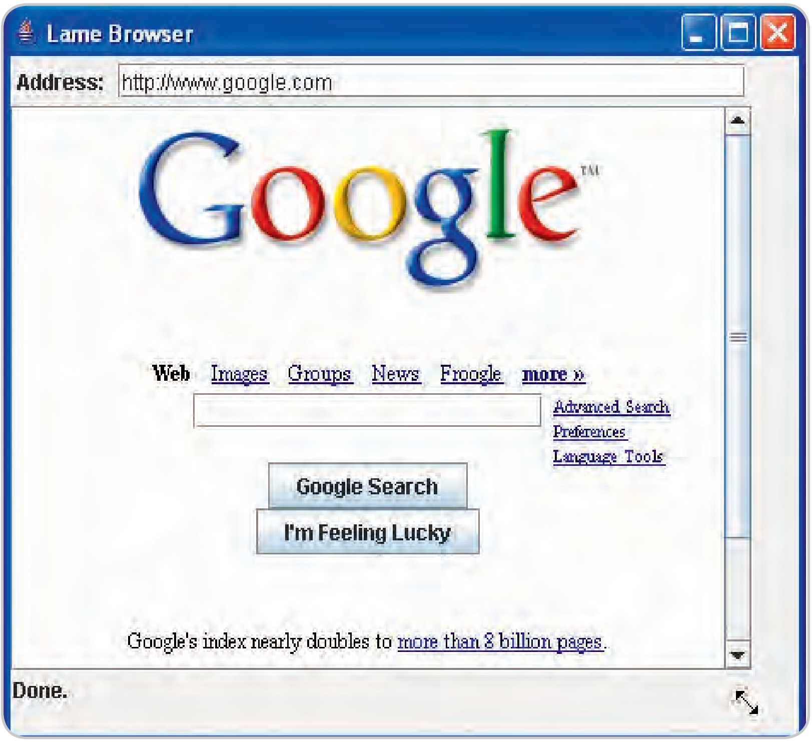
The trick to doing this in Java is to use a Timer to periodically check the current bounds of the window (Frame), and if it has changed since the last interval, send the validate/repaint messages to the Frame. You can see how this is done in Listing 5. I coded it as a static method in FrameResizer so this functionality can be used independently of the component.

The code uses a static HashMap (continuousLayoutWindows) so it can be used for multiple windows in a multi-frame application. It lazily initializes this HashMap. In the map it stores the current Container bounds keyed by the Container. If the current bounds are different than what is in the map, the current bounds are saved and the validate and repaint messages are sent to the Container (our static validate() method from before). That's all there is to it. I use 100 milliseconds (hard-coded) as my timer interval – feel free to experiment with this number (or parameterize it).

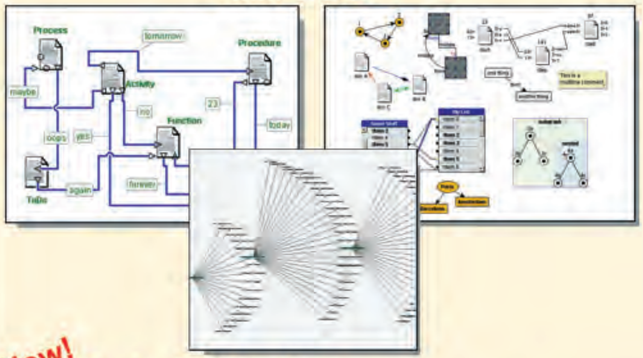
You can specify whether or not "continuous layout" is used by the FrameResizer in its constructor (the default is true). Note that one drawback to "continuous layout" in this context is the flicker that occurs during the repaint. This is unfortunately unavoidable. Also note that the continuous layout behavior is not required if you are using the FrameResizer in the context of a JInternalFrame.

Summary

In this article I presented a custom Swing component that can be used to help provide a customary look and feel to a Frame or JInternalFrame. I've also introduced the concept of "continuous layout" to top-level Java desktop windows and shown how this can be implemented. The source code for this article can be downloaded from http://jdj.sys-con.com.


Lame Browser
Address: http://www.google.com
Google
Web Images Groups News Froogle more »
Advanced Search
Preferences
Language Tools
Google Search
I'm Feeling Lucky
Google's index nearly doubles to more than 8 billion pages.
Done.
Figure 3 Browser window

Build Incredible Interactive Diagrams with JGo™



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

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

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

Northwoods SOFTWARE CORPORATION

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

Listing 1

```
public void paintComponent(Graphics g) {
    super.paintComponent(g);
    Dimension size = getSize();
    Insets insets = getInsets();

    int x = size.width - insets.left - insets.right - 2;
    int y = size.height - insets.top - insets.bottom - 2;

    for (int i = 0; i <= 2; i++) {  // paint ribs, lower
     right up
  paintRib(g, x, y - (i * 4));
    }

    for (int i = 0; i <= 2; i++) {  // paint ribs, lower
      right left
        if (i != 0) {
      paintRib(g, x - (i * 4), y);
  }
    }

    paintRib(g, x - 4, y - 4);  // paint last rib, lower
right back
}
```

Listing 2

```
public void mousePressed(MouseEvent e) {
    Point pt = new Point(e.getPoint());
    SwingUtilities.convertPointToScreen(pt, FrameResizer.this);

    Rectangle frameBounds = parentContainer.getBounds();

    mouseAdjust.x = frameBounds.x + frameBounds.width -
      pt.x - 1;
    mouseAdjust.y = frameBounds.y + frameBounds.height
- pt.y - 1;
}
```

Listing 3

```
public void mouseDragged(MouseEvent e) {
    mouseLocation.x = e.getX();
    mouseLocation.y = e.getY();

    SwingUtilities.convertPointToScreen(mouseLocation,
                                        FrameResizer.this);

    Rectangle frameBounds = parentContainer.getBounds();
    parentContainer.setBounds(frameBounds.x, frameBounds.y,
mouseLocation.x - frameBounds.x + mouseAdjust.x,
  mouseLocation.y - frameBounds.y + mouseAdjust.y);
}
```

Listing 4

```
private static void validate(final Container container) {
    SwingUtilities.invokeLater(new Runnable() {
   public void run() {
  if (container instanceof Frame) {
           ((Frame)container).invalidate();
      ((Frame)container).validate();
  } else if (container instanceof JInternalFrame) {
                ((JInternalFrame)container).
                  revalidate();
      ((JInternalFrame)container).repaint();
  }
  container.repaint();
       }
    });
}
```

Listing 5

```
public static void setContinuousLayout(final Container
container) {
    new Timer(100, new ActionListener() {
        public void actionPerformed(ActionEvent e) {
       Dimension currentSize = container.getSize();
  if (continuousLayoutWindows == null) {
      continuousLayoutWindows = new HashMap();
  }
  Dimension windowSize =
(Dimension)continuousLayoutWindows.get(container);
  if (windowSize == null) {
                windowSize = new Dimension();
                continuousLayoutWindows.put(container,
                                          windowSize);
  }
  if (!currentSize.equals(windowSize)) {
                windowSize.width = currentSize.width;
                windowSize.height = currentSize.height;
                validate(container);
  }
        }
    }).start();
}
```

Reach Over 100,000

Enterprise Development Managers & Decision Makers with...

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

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

SPRING 2005

IT solutions >GUIDE

www.sys-con.com/IT

Develop Locally Run Globally

Tools and Solutions For the World Class Enterprise

Top IT Solution Providers!

- > ClearNova
- > EV1 Servers
- > Extentech
- > GraphOn
- > Hosting.com
- > Synaptris
- > InterSystems
- > JadeLiquid Software
- > MapInfo
- > Parasoft
- > PDF Tools AG
- > Quest Software
- > Verio
- > Visual Paradigm

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

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

Get Listed as a Top 20* Solutions Provider

For Advertising Details Call 201 802-3021 Today!

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

SYS-CON MEDIA

The World's Leading i-Technology Publisher

Interface All Boundaries

Providing tangible benefits to your application

by Pete Whitney

Experienced developers know many of the benefits of and motivations for using interface-based design principles. Interfaces provide for polymorphic behavior by hiding the implementation and only exposing the relevant public methods of the implementing class. What may be less appreciated is that the use of interfaces, when applied to an entire application, can provide for application isolation, while at the same time enhancing testing capabilities.

In this article we'll explore the use of interfaces at all application boundaries and gain an appreciation for why we should consider, and possibly even mandate, interface-based design principles at all application boundaries. This mandate should apply even if application requirements or application design do not call for differing behavior behind the interface.

What Is an Application Boundary?

Before we start mandating anything throughout our entire application, it would be helpful to have a better feel for exactly what an application boundary is. Consider any application you write. The application is composed of the classes that you write and the code or resources with which your application interacts. Unless you're writing an "Hello World" application, you will need to access many or all of the following resources:

- File system, library, server, database…

Although the above list is only a small subset of the many external resources that should be considered application boundary resources, it does convey the essence of the idea. Figure 1 provides a pictorial view of these application boundary resources.

Motivations for Interfaces

While many motivations drive the use of interfaces, it is instructive to describe the core motivations. As one example, consider JDBC, which makes heavy use of interface-based design principles. A ResultSet is an interface; moreover, all JDBC driver vendors must provide an implementation for the ResultSet interface. Application developers simply use the ResultSet, in most instances not knowing or caring how the underlying code is implemented. Application developers code to the ResultSet interface while the actual implementation is usually defined and configured apart from the application code. JDBC is based heavily on interfaces to support the ability to change the driver implementation without impacting the application code. Simply put, portability is the driving factor for JDBC interfaces.

Resource wrapping can be listed as a second motivation for the use of interfaces. The main intent is to support the ability to change the wrapped resource with little to no application code changes. In this sense, the motivations for resource wrapping are very similar to the JDBC example except that most resources are not based on a set of predefined interfaces as JDBC is. By contrast, different resources that deliver the same or similar functionality are not likely to have the same API. As a result, the interface for a changed resource is likely to become more of a façade-based implementation, especially if a resource change is realized.

Other application interfaces are typically chosen for the polymorphic benefits realized by an interface-based design. The application behavior is mandated by either specific requirements or by design recognizing the similarities between two or more subsystems or elements. On the application side, interface-based solutions are nearly always chosen based on required or desired application behavior with polymorphism being the driving motivation.

Lastly, application isolation should be added as another motivation for the use of interfaces. The intent of application isolation is not necessarily to support the future change of a tool or resource, but rather to support the ability to remove the resource dependency from an application. Isolation is often needed during unit testing, but is also beneficial in the

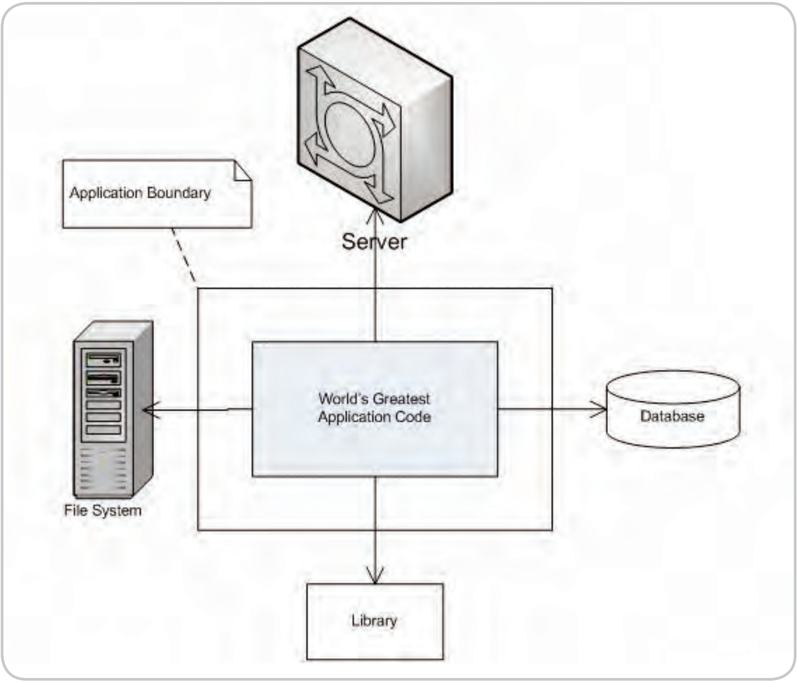


Figure 1

Pete Whitney is a principle engineer at Nexa Technologies Inc. in Allen, TX. Pete received his BS in computer science from the University of Texas at Dallas. His industry interests are in genetic algorithms for business, generic programming, and application architecture.

pnwhitney@sbcglobal.net

early stages of development when the resource in question may not yet be available. Using interfaces at all application boundaries delivers the ability to easily plug in alternative implementations that are not resource dependant. This capability can deliver extensive benefits to a development team by eliminating the constraints imposed by some resource dependencies.

For brevity, the motivations for interface-based design are:

- Portability
- Resource wrapping
- Polymorphism
- Application isolation

Project Example, Iteration 1

Application isolation is the motivation we'll spend the remainder of this article on. With that thought in mind, let's turn to the impact of external resources on our application code base. For better understanding we'll use the retrieval of a collection of person objects from a database as our core example. In iteration 1 we'll provide an unimproved implementation. This iteration shows a small cut of the code as it might exist without the use of interface-based design principles. Using good design principles we'll encapsulate our database access by using a PersonDAO class (*D*ata *A*ccess *O*bject pattern) for all database access. Figure 2 shows the UML for our simplified application.

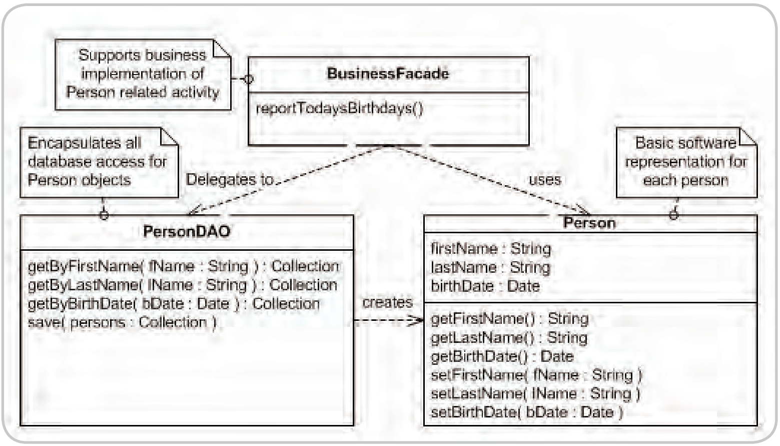
The PersonDAO class will acquire a database connection and issue queries on behalf of the invoking party, which in this instance is the BusinessFacade class (Façade Pattern). The code below shows a brief implementation of the BusinessFacade class. Note the direct instantiation and use of the PersonDAO class, which immediately ties the BusinessFacade class to the database resources referenced in the PersonDAO class.

```
public class BusinessFacade
{
  public void
       reportTodaysBirthdays()
  {
    Date today = new Date();
    PersonDAO personDao =
       new PersonDAO();
    Collection persons =
     personDao.getByBirthDate(today);
    // Do what needs to be done
    //  to report on those that
    //  have birthdays today.
  }
}
```

Being tied to the database imposes all of the following resource requirements:

- Existing network connectivity
- Database server capable of handling the connection request
- Database schema supporting the Person table
- Database data from which the query can be fulfilled

It's certainly true that we will be required to be tied to some database at some time during the development phase. However, imposing such a hard dependency for all development unnecessarily hinders other developers working outside the realm of the database. Moreover, the hard dependency can be rather easily avoided with a little forethought and a simple interface!



Supports business implementation of Person related activity
BusinessFacade
reportTodaysBirthdays()
Encapsulates all database access for Person objects
Delegates to
uses
Basic software representation for each person
PersonDAO
getByFirstName(fName : String) : Collection
getByLastName(lName : String) : Collection
getByBirthDate(bDate : Date) : Collection
save(persons : Collection)
creates
Person
firstName : String
lastName : String
birthDate : Date
getFirstName() : String
getLastName() : String
getBirthDate() : Date
setFirstName(fName : String)
setLastName(lName : String)
setBirthDate(bDate : Date)

Figure 2

Project Example, Iteration 2

In iteration 2 we seek to avoid the database dependency imposed by our thoughtless implementation in iteration 1. Adding an interface and a few support classes is all that's needed to remove our database dependency. Figure 3 shows the UML for our example application with the requisite changes.

Note the specification of the IPersonDAO interface. Implementing this interface are two classes namely "PersonDAO" and "TestPersonDAO". The PersonDAO class may remain as it was before except for designating that the class now implements the IPersonDAO interface. The TestPersonDAO class

WEEKEND WITH EXPERTS
Get a Java injection on the go!
Spend a weekend and have a lunch with experts in an intimate learning environment in the city near you.
No salesmen allowed. Join our technical discussions and hands-on workshops.
The next Roadshow is in New York City on October 15 and 16.
www.weekendwithexperts.com

is created to return a statically defined set of Person instances in a collection. By doing so, the TestPersonDAO class will have no actual dependency on the database, but can function in place of the database-constrained PersonDAO class. The following provides the code implementation of the TestPersonDAO class:

```
public class TestPersonDAO
        implements IPersonDAO
{
  public Collection
        getByBirthDate( Date bDate )
  {
    ArrayList results =
        new ArrayList();
    Person person = new Person();
    // set person attributes here
    results.add( person );

    // Construct more persons as
    // needed and add these to
    // the returned results
    return results;
  }
}
```

The database dependency is removed from the BusinessFacade class when providing it with a TestPersonDAO instance rather than a PersonDAO instance. This functionality is provided for via delegation to the DAOFactory class:

```
public class BusinessFacade
{
  public void
          reportTodaysBirthdays()
  {
    Date today = new Date();
    IPersonDAO personDao =
     DAOFactory.createPersonDAO();
    Collection persons =
     personDao.getByBirthDate(today);
    // Do whatever needs to be done
    // to report on those that have
    // birthdays today.

  }
}
```

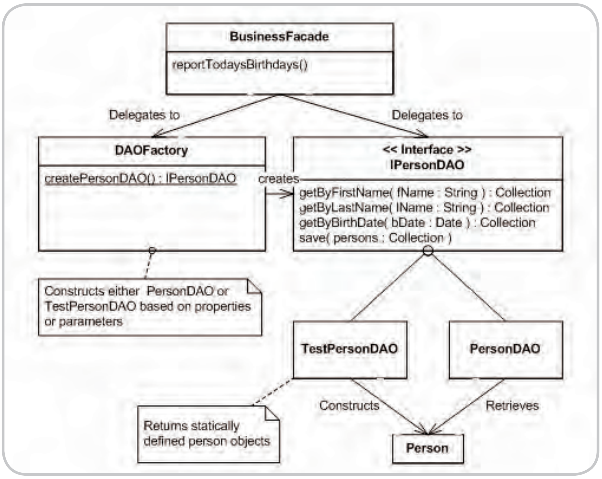

BusinessFacade
reportTodaysBirthdays()
Delegates to
Delegates to
DAOFactory
createPersonDAO() : IPersonDAO
creates
<< Interface >>
IPersonDAO
getByFirstName(fName : String) : Collection
getByLastName(lName : String) : Collection
getByBirthDate(bDate : Date) : Collection
save(persons : Collection)
Constructs either PersonDAO or TestPersonDAO based on properties or parameters
TestPersonDAO
PersonDAO
Constructs
Retrieves
Returns statically defined person objects
Person

Figure 3

Many possibilities exist for the implementation of the DAOFactory class. The different class instantiations could be defined by properties, the creation method could be passed a parameter, or the implementation could be based on the AbstractFactory pattern. Regardless of the chosen implementation, the important elements are as follows:

- The BusinessFacade implementation invokes upon an interface instance rather than a concrete instance that it creates.
- The BusinessFacade implementation delegates the construction of the IPersonDAO instance to some other entity (DAOFactory).

By applying these changes, the BusinessFacade class will only be database coupled during those times that the DAOFactory returns a PersonDAO instance rather than a TestPersonDAO instance. Defining the actual construction and return type via configuration rather than code will allow the database dependency to be removed via the same configuration.

Other Tools

One tool that exists for the purpose of removing application dependencies is mock objects. The mock objects API allows a developer to easily instantiate objects based on any interface, and to easily provide a default implementation for these mocked up objects. The mock object framework shines in the area of unit testing by allowing dependant resources to be "mocked" in a programmatic way. As such a developer can create defined software instances for required resources and allow tests to be driven from the mocked instances. This greatly supports the isolated "sand-box" environment that is so desirable for well-written unit tests.

It should be noted that the interface-based design principles proposed in this article will lay the groundwork for your application to easily incorporate and leverage the use of the mock objects framework. Whether you provide your own implementations or leverage the mock objects framework, you will have allowed your application to be configured apart from the resources that it would otherwise depend upon.

Conclusion

We've had the opportunity to explore the problems that arise when direct resource dependencies are imposed upon our application development team. The benefits of avoiding these dependencies offer compelling reasons to avoid or defer such dependencies. We've also experienced the relative ease in which these dependencies can be avoided.

In much of my development life, I've defined resource interfaces initially, provided non-dependant implementations for these interfaces, and went well along the early iteration phase or phases without ever needing to invoke upon the implementation that introduces the actual resource. As a result, my early phase development goes faster because I'm not hindered by the absence of a dependant resource. Only later in the development phase do I introduce the implementation that adds the resource dependency.

Requiring that all external resources be based on an interface adds little effort to the development timeline. Yet it provides tangible benefits to the application under development and to the application development team. Therefore, consider adding application isolation to your list of motivations when considering the use of interfaces in your application development efforts and use interfaces at all application boundaries.

As a final note, it is instructive to recognize that the techniques described in this article transcend the implementation language. Although the code examples were presented in Java, the benefits of application boundary interfaces can be realized in any implementation language.

References

- Alur, D. et al. (2001). *Core J2EE Patterns*. Prentice Hall.
- Gamma, E., et al. (1995). *Design Patterns*. Addison-Wesley.
- *Mock Objects:* www.mockobjects.com

JAVA DEVELOPER'S JOURNAL Advertiser **Index**

Advertiser	URL	Phone	Page
Altova	www.altova.com	978-816-1600	4
ceTe Software	www.dynamicpdf.com	800-631-5006	45
Common Controls	www.common-controls.com	+49 (0) 6151/13 6 31-0	49
Google	www.google.com/jdj	650-253-0000	41
InetSoft	www.inetsoft.com/jdj	888-216-2353	23
Information Storage & Security Journal	www.issjournal.com	888-303-5282	59
InterSystems	www.intersystems.com/cache14p	617-621-0600	Cover IV
IT Solutions Guide	www.sys-con.com/it	888-303-5282	55
JadeLiquid Software	www.webrenderer.com	+61 3 6226 6274	25
Java Developer's Journal	www.jdj.sys-con.com	888-303-5282	47
Jinfonet Software	www.jinfonet.com/jp6	301-838-5560	31
LinuxWorld Conference & Expo	www.linuxworldexpo.com	800-657-1474	33
M7	www.m7.com/power	866-770-9770	19
Microsoft	microsoft.com/connectedsystems		Cover II
NCL	www.nclt.com/jdj	+353 1 6761144	39
Northwoods Software Corp.	www.nwoods.com/go	800-434-9820	53
Parasoft Corporation	www.parasoft.com/jdjmagazine	888-305-0041	7
Parasoft Corporation	www.parasoft.com/soa_wsj	888-305-0041	29
Perforce Software	www.perforce.com	510-864-7400	11
ReportingEngines	www.reportingengines.com	888-884-8665	15
Smart Data Processing, Inc.	www.weekendwithexperts.com	732-598-4027	57
Software FX	www.softwarefx.com	800-392-4278	Cover III
STP Conference	www.stpcon.com	415-785-3419	37
Synaptris	www.intelliview.com/jdj	866-99IVIEW	17
SYS-CON Website	www.sys-con.com	888-303-5282	51

Magic Is Golden

Pulling rabbits out of hats

by Daniel Brookshier

Magic, like software, depends on understanding the audience. Why not use a few techniques from magic to understand users?

A Golden Hammer Appears

First, let's explore the magic of making objects appear out of thin air. I'm thinking big, so let's make a house appear. Is the magician building the house? No, just revealing it. Do you care how the house got there? No, it's just cool to have the house. Software should be just as magical. What's more magical, a feature-rich virus scanner with lots of options to set with the capacity to eliminate 98% of spam, or a simple scanner that eliminates 97% of spam? Read on.

How does magic and mind reading enter into this? Before I pull a rabbit out of my hat, we need to be able to spot a lack of magic. I use the Golden Hammer anti pattern from my favorite software book called *Anti Patterns - Refactoring Software, Architectures, and Projects in Crisis* by William J. Brown, et al. The Golden Hammer anti pattern applies to any application that is meant to be a tool or set of tools rather than a carefully constructed solution for a specific problem.

The Golden Hammer anti pattern is easily recognized. We just look for a few key words and phrases like framework, scripting, kernel, engine, customer-driven, or user-scripted. Related warning signs are multi-day training programs, 24-hour premium support, or the presence of thick manuals. Things you don't hear are instant-on, works out of the box, or solution driven.

A Golden Hammer application is so flexible it can never fail. That is a seductive thought. We create the perfect software by "not" solving a problem. Give the user the tools and let him build his own house to live in. What could be simpler?

The problem is that tools are not solutions. It's the difference between a house and a bunch of tools and wood. It's also the difference between an application that solves a problem now and one that might take a week of training plus trial and error.

But this article is about magic. Magic's first principal is to hide the trick, but the key is to control the audience so they see the illusion. Magicians create, control, and present illusions in the same way application designers create well-designed applications. The less you expose control of an application and match the user's process, the more predictable, testable, and easily used your applications will become.

Is This Your Card?

Let's talk about the core of many card tricks, the card force. The common scenario is that you are asked to pick a card, any card, memorize it, and then place it back into the deck. The magician then reproduces the card, perhaps spray painted on the side of an elephant that appears in a flash of fire. The magician's goal is to perform this trick without you realizing that he is forcing you to select a card already on the elephant. There are other variations, but forcing is tried and true with thousands of variations.

How does forcing a card relate to software? We want the user to appear to be in control. If a magician just selected a card, showed it to the audience, put it back into the deck, shuffled, and reproduced the card, we are not impressed. But if you get to select the card, then shuffle it into the deck, we have the illusion of control and thus the production of the card is magic.

How do we select the right card for the user? Reading the minds of many of you I see requirements and process. The core component I want you to concentrate on is the process and the solution. What do they do now? What will save them the most time? What is required every time they perform the task? What is the process flow? What is wrong with the current solution?

Creating the magic is as simple as using this information to constrain the design to just solve the problem in a constrained way. The forcing of the card, like limiting the choices a user can make, are key to the success of the total application. Flexibility remains, but it is constrained to a solution and its highly probable variations.

Your gauge to success is testability. But not just if A then B. Your best tests will be problem/solution or goal-oriented tests. Simply, can the user solve the problem quickly and easily out of the box? That's real magic.

Think of a Number, Any Number

Mentalism, or mind reading, is a variation of the magical arts. A mentalist either reads your mind or makes predictions about past and future. Many mentalist tricks use human behavior and a few statistics and observations. The mentalist appears to read your mind but really they are using clues about your life, a little misdirection, and some common rules of thumb.

Daniel Brookshier, JXTA consultant, is a freelance consultant, speaker, author, and Java geek since Java 1.0. He is one of the core members at jxta.org and runs several open source projects including jxta-remote-desktop at java.net. Daniel's latest book is *JXTA: Java P2P Programming*, but he also writes articles for java.sun.com and *P2PJournal* where he is an editor. Daniel's blog covers the P2P world, tips, tricks, and musings on Java and JXTA.

turbogeek@cluck.com

"The less you expose control of an application and match the user's process, the more predictable, testable, and easily used your applications will become"

Pick a number; say between 1 and 5…. If you are like 95% of most of my audiences, you will pick three. The numbers four, two, one, and five follows that. Even if you say you picked another number, 3 was what you first thought. It's a cool trick but just shows how easy it is to use human behavior to help write software.

I Predict Great Wealth

At one company, we actually hired a mentalist to help us write consumer-profiling software. Our core problem was avoiding a complex interface to gather personal information. Dan Korem (www.ifpinc.com) was hired as a specialist on profiling (including work with the FBI). But surprisingly Dan started out as a stage magician and mentalist and a core part of his profiling skills that we needed were from his stage act. His advice to us was to use the techniques of the mind reader and specifically an art known as Cold Reading.

Cold Reading starts with a few bits of information to create new assumptions about a person and their life. As an example, the mentalist (or fake telephone/TV psychics) starts with a person's age and the presence of a wedding ring to guess with high probability that the person has children. The mentalist can then make further assumptions that the parent has thoughts of angst about their children doing well in school. A+B implies a high probability of C, which then assumes D. There is a bit more to this, but that's the core of appearing to being a mind reader and even appearing to predict the future or looking to your past.

Using the techniques of the mentalist we created an accurate picture that was corrected over time based on observing user behavior. We just used basic demographics like age, profession, and a couple other facts to make a large number of assumptions. We didn't need to expose the user to page after page of questions. The design of our profile interface became mostly invisible even though it was far more powerful.

Is This the File You Lost?

Mentalist tricks can also get you one step ahead of the user by using context to predict actions. You can see signs of this in applications that collapse "find" and 'find and replace" in their menus. Instead the find dialog anticipates by integrating "replace" and "find." Or "export" is integrated with "save as" because saving a file with a new name or location is a similar process to changing its format.

If I mention children, you'll think of your own. If I say that you are worried about their safety, I am dead on with that prediction. There are many such contexts with common thoughts and motivations that we all share. We don't tell the mind reader and yet they know.

Let's apply context to saving backups of files you are editing. My application looks like it has psychic powers after the computer crashes if I show you a file saved five seconds before the crash. No need to bother you with the option for backups. The experience after the crash is important. Before the crash, I might want backups, but after a crash I definitely want them. Giving you the option of turning on and off backups is meaningless and should not be a part of the application.

What about the interval between backups and saves? Instead of letting the user guess the interval, why not watch what the user is doing? If I am really typing away, I want to do more saves. If I'm idle, why save at all? Context drives the solution and real-time observation is even better. Perhaps more coding, but the result is a simpler interface and a much happier user when the backup file magically appears.

The Magic of Great Software Design

With software we can have miracles all the time. Users don't want to see the wires, trapdoors, or mirrors. We know magic is trickery, but if we can't see the wires, we react to the miracle. Forcing flow, profiling, and using context reduces user complexity and adds more value.

Back to the ideas of tools verses solutions. The audience does not want to build the magician's props or feed the pigeons and rabbits. Users are quite happy with picking the forced card and being entertained with magic.

Want to give a user a Golden Hammer instead of solutions? Hey, give them an editor and teach them to write Java code. To me, that's too much like a magician revealing his tricks!

Users are experts in their domain. They are not magicians or programmers. Respect their time and skills and don't waste their time learning your profession. Instead learn theirs like a magician studies what will fool and amaze you.

I Believe in Magic

Not everyone likes magic. To some it is abhorrent that anyone would even be entertained by an illusion. I can look deep into the minds of these rebels of pure logic. You who believe flexibility "is" efficiency. I don't argue with your methods or philosophy. You are 100% correct. Flexibility has a payoff. But please, most users don't want that kind of flexibility. Users will be more efficient within their domain, skills, and desires. Most people don't want to see what's behind the curtain or operate the levers you love. Please respect that.

I do believe in magic. Magic is anything where I get what I need without a need to know how it is done or, in fact, the skills to do it myself. The same is true of my software and the software I design for my clients. I'm a programmer and magician and I have users and an audience. My users believe in magic. To bring back the example of the hard to use versus the easy virus scanner, the easier one to use has all the magic. Abracadabra!

JCP Launches **New Program**

First constellation of Star Spec Leads takes shape

by Onno Kluyt

The JavaOne Conference was the stage for many Java premiers and launches. One of them was the Star Spec Leads program initiated by the JCP.

This program is the community's way of recognizing and celebrating those Spec Leads who had an exceptional contribution to the development of Java Specification Requests (JSR) and carried out their spec lead duties flawlessly. Specification leads, in widely adopted JCP jargon Spec Leads, are instrumental to the development and finalization of Java standards. Without them the proposed specifications wouldn't go beyond the idea stage.

Star spec leads in the definition of the Star Spec Leads program produce high-quality specifications, establish best practices, and mentor entry spec leads. The program endorses the good work that they do and showcases their methods for other spec leads to emulate.

It's no easy thing to become a Star Spec Lead. You need a proven track record of successful communication with the expert group and the PMO. You need to respond promptly to concerns the expert group, PMO, or the ECs may have; conduct expert group meetings effectively and share JSR comments on a regular basis; show leadership and inspire and motivate the expert group; demonstrate rigorous discipline by staying on schedule and delivering JSRs within expectations and conducting JSR business in an open, transparent manner.

The first ones to cut this ambitious profile were announced by the JCP at the launch of the program the first day of JavaOne. Based on results of a poll of Expert Group (EG) members, Executive Committee (EC) members, and PMO staff combined, 16 spec leads won the distinction of Star Spec Lead: Alejandro Abdelnur of Sun, Volker Bauche of Siemens, John Buford of Panasonic and Network Technologies Laboratories, Ekaterina Chtcherbina of Siemens, Linda DeMichiel of Sun, Andreas Ebbert of Nokia, Jan Eichholz of Siemens, Stefan Hepper of IBM, Mark Hornick of Oracle, Jere Kapyaho of Nokia, Kimmo Loytana of Nokia, Eamonn McManus of Sun, David Nuescheler of Day Software, Eric Overtoom of Motorola, Vincent Perrot of Sun, and Jim Van Peursem of Motorola.

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

onno@jcp.org

Due to space constraints, I'll only be able to introduce a few of them to you this month.

Volker Bauche is a senior software technologist and team lead in the Siemens Mobile software technology department. He stays on the cutting edge, currently focusing on predevelopment projects, evaluations, prototypes, and so forth, which often incorporate Java technology. He got involved in the JCP work as early as 1998 when he became part of the Siemens team working on the Java Platform, Micro Edition standardization process. His JCP portfolio includes participation and contributions to JSR 118 MIDP 2.0; JSR 179 Location API for Java Platform, Micro Edition (Java ME); JSR 238 Mobile Internationalization API; JSR 256 Mobile Sensor API; and JSR 257 Contactless Communication API. Volker also served as co-Spec Lead with Jari Länsiö of Nokia on JSR 195 Information Module Profile (IMP) and JSR 228 IMP – Next Generation.

As a lead scientist for Panasonic Digital Networking Lab, ***John Buford*** conducts research in middleware for Java technology-enabled consumer electronics devices. John has worked with Java technology from the perspective of a developer, project lead, vendor, and researcher, covering all three editions of the Java Platform. He has developed clients for multimedia/hypermedia servers, network management systems, and workflow/e-commerce systems. He was the first to use Java Platform, Enterprise Edition (Java EE) application servers for telecommunications network management and the first to implement the Java Virtual Machine Profiler Interface (JVMPI) for a Mobile Information Device Profile/Connected Limited Device Configuration (MIDP/CLDC) platform. John participated in developing a Java ME VM for various mobile and embedded platforms, including a Just-in-Time (JIT) compiler for the MIDP/CLDC platform. Currently John serves as a secondary representative to the EC, Expert Group member of JSR 259 Ad Hoc Networking API, and co-Spec Lead on four JSRs: JSR 164 SIMPLE Presence, JSR 165 SIMPLE Instant Messaging, JSR 186 Presence, and JSR 187 Instant Messaging

Andreas Ebbert, a software design engineer at Nokia, is responsible for the Java Platform, Enterprise Edition (Java EE) programming for the Nokia NetAct network and service management system. His participation in the JCP program began in 2001 when Andreas joined the OSS through Java Initiative (OSS/J). He created the Reference Implementation (RI) and part of the Technology Compatibility Kit (TCK) for JSR 89 OSS Service Activation API. He participated in all OSS JSRs and in the development of the RI for JSR 144 OSS Common API. He serves as an Expert for JSR 263 Fault Management API and is Spec Lead for JSR 264 OrderManagement API.

A mathematician by education, ***Jan Eichholz*** of Siemens first participated in the standardization of the Mobile Information Device Profile (MIDP) in 1999. Since then, Jan has participated as an Expert Group member in several Java Specification Requests (JSRs), all related to enhancing wireless technology and also served as Spec Lead for JSR 120 Wireless Messaging API (WMA) and JSR 205 WMA 2.0.

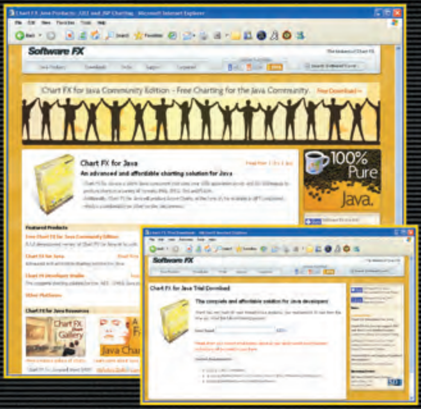
A software architect with IBM, ***Stefan Hepper*** started programming with Java technology when it was still at version 0.9, circa 1996. He began participating in the JCP program at the beginning of 2002 because customers wanted something done. He became co-spec lead with Sun's Alejandro Abdelnur for JSR 168 Portlet Specification, a standard enabling interoperability between portlets and portals. On behalf of IBM Stefan assumed responsibility for creating the Reference Implementation (RI) and on behalf of Sun Alejandro created the Technology Compatibility Kit (TCK).

What do all these folks have in common? They are passionate about Java, and they believe in the benefits of evolving the platform based on binary standards that ensure compatibility and thereby prevent problems that many customers would otherwise encounter.

Stay tuned for more Star Spec Leads profiles next month and check out more details about the Star Spec Leads program and their success with developing standards for the Java platform by visiting http://jcp.org/en/press/news/star.

Software FX

Zero To Chart In Less Than An Hour!



9:04 am

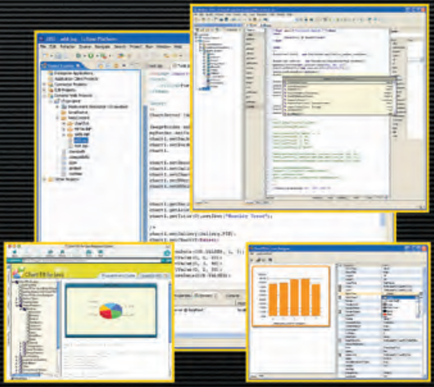
To learn more about the Chart FX products visit www.softwarefx.com and decide which one is right for your platform and target. Then download the 30-day trial version or the Chart FX for Java Community Edition which is a FREE, non-expiring, full development version.

9:07 am

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

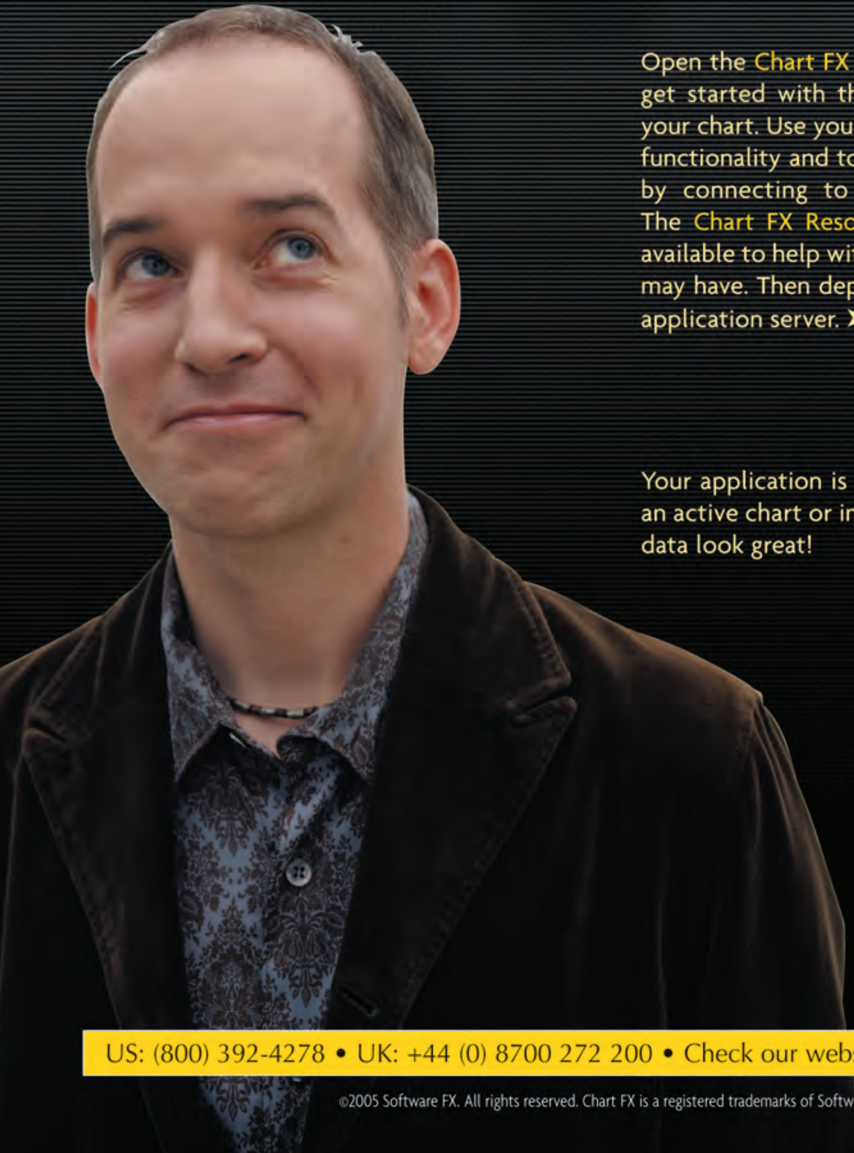
9:13 am

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



9:58 am

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



Ready... Set... Download!

Chart FX

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

Innovations by InterSystems

Rapid development with robust objects

Lightning speed with a multidimensional engine

Easy database administration

Massive scalability on minimal hardware

No More Object-Relational Mapping.

Caché is the first multidimensional database for transaction processing and real-time analytics. Its post-relational technology combines robust objects and robust SQL, thus eliminating object-relational mapping. It delivers massive scalability on minimal hardware, requires little administration, and incorporates a rapid application development environment.

These innovations mean faster time-to-market, lower cost of operations, and higher application performance. We back these claims with this money-back guarantee: *Buy Caché for new application development, and for up to one year you can return the license for a full refund if you are unhappy for any reason.** Caché is available for Unix, Linux, Windows, Mac OS X, and OpenVMS – and it's deployed on more than 100,000 systems ranging from two to over 50,000 users. We are InterSystems, a global software company with a track record of innovation for more than 25 years.

InterSystems CACHÉ™

Try an innovative database for free: Download a fully functional, non-expiring copy of Caché, or request it on CD, at **www.InterSystems.com/Cache14P**

* Read about our money-back guarantee at the web page shown above.