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

JDJ

JDJ.SYS-CON.COM VOL.11 ISSUE:12

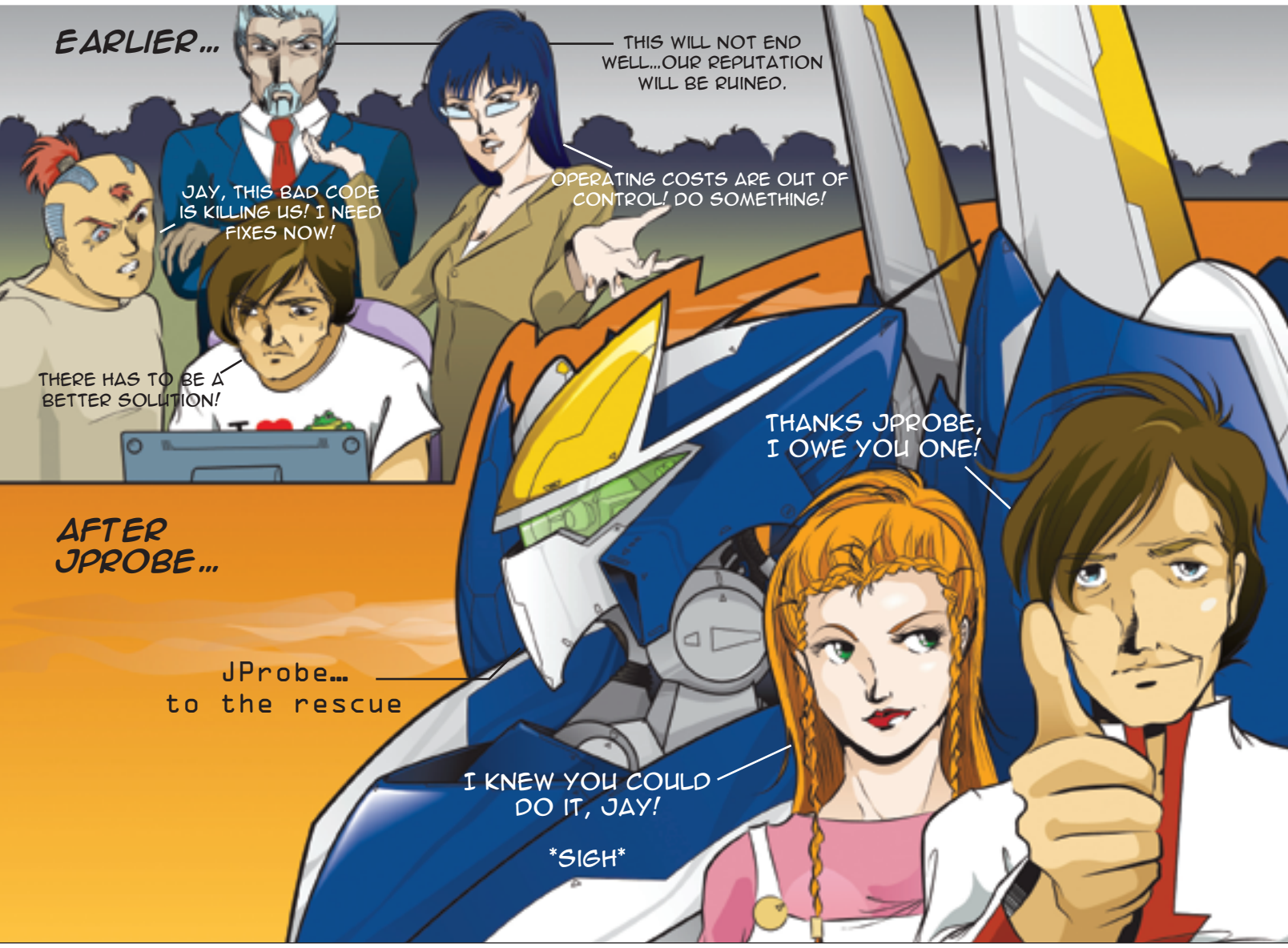
CONCURRENT QUERIES

A pattern for improving database query performance

RETAILERS PLEASE DISPLAY
UNTIL FEBRUARY 28, 2007
$5.99US $6.99CAN
01>
0 09281 01751 6

PLUS...

- Is Your Mobile Safe?
- The Development Power of Open Source AJAX Tooling
- j-Interop: An Open Source Library for COM Interoperability Without JNI



* Free t-shirt offer

In the Battle Against Bad Java Code…

JProbe® is on your side.

Jay, our Development Manager, was in trouble. He had the Application Business Owner, Production IT Manager and even the CIO on his back about bad code finding its way into production. Operation costs were skyrocketing, productivity was down and customer loyalty was at risk.

Then Jay discovered JProbe® from Quest Software. With JProbe, Jay's team can proactively zero in on the code that affects performance – before the code goes to production. Now Jay is a hero and delivers quality, high-performing code on time – every time.

Join the battle against bad Java code in production. Download a trial of JProbe at www.quest.com/hero — and get a free JProbe t-shirt!*

QUEST SOFTWARE®

JAVA DEVELOPER'S JOURNAL

Editorial Offices
SYS-CON Media, 577 Chestnut Ridge Rd., Woodcliff Lake, NJ 07677
Telephone: 201 802-3000 Fax: 201 782-9638

Java Developer's Journal (ISSN#1087-6944) is published monthly (12 times a year) for $69.99 by SYS-CON Publications, Inc., 577 Chestnut Ridge Road, Woodcliff Lake, NJ 07677. Periodicals postage rates are paid at Woodcliff Lake, NJ 07677 and additional mailing offices. Postmaster: Send address changes to: Java Developer's Journal, SYS-CON Publications, Inc., 577 Chestnut Ridge Road, Woodcliff Lake, NJ 07677.

Is the Rise of Google the End of the Game for Everyone Else?

Jeremy Geelan

As I write this, the stock price of Google, Inc. just exceeded $500 for the first time in the company's still-brief (two-year) history as a public company. That gives the search colossus a market cap of $150 billion, many times in excess of its physical assets – currently valued at $10.2 billion.

Whether the latest surge in value is being driven by the perception that Microsoft may be losing its golden touch, or whether it is Google's sheer Web 2.0-style inventiveness that is causing investors to pile into its stock, matters not. What matters is that the company that snapped up video-sharing site YouTube for $1.65 billion now doesn't seem quite so profligate. Everything is relative.

But why, many outside the industry are wondering, is the company started eight years ago in a Silicon Valley garage by Stanford University graduate students Larry Page and Sergey Brin already worth $150 billion, when the one started 24 years ago by Andy Bechtolsheim, Bill Joy, Vinod Khosla, and Scott McNealy – a.k.a. **S**tanford **U**niversity **N**etwork, now Sun Micrososystems – is currently worth just $19.5 billion?

The answer, ironically, may lie in Andy Bechtolsheim. Because not only is he famous for being Sun's "employee No. 1," he is also equally famous for being the author of a $100,000 check that represented nearly one-tenth of Google's total capital when it was founded, back in 1998 when it was still running off the google.stanford.edu domain – in other words, the Stanford University website.

Although Bechtolsheim rejoined Sun in February 2004 when it acquired the privately-held company he co-founded, Kealia, based in Palo Alto, California, Sun's first "disruptive innovator" is uniquely independent in spirit. What he saw in Google back then, long before it became The Big G, while very different from what we see today, must have captivated him: their front end of public search and advertising algorithms, he must have realized, had unusual and disruptive potential.

Just five years later, another Sun employee, Eric Schmidt, experienced a similar epiphany. In a vision famously summarized later as "The Network Is the Computer," Schmidt wrote: "When the network becomes as fast as the processor, the computer hollows out and spreads across the network." Under such new circumstances, Schmidt figured, profits would flow very differently:

> "Not to the companies making the fastest processors or best operating systems, *but to the companies with the best networks and the best search and sort algorithms.*"

Schmidt left Sun and, as we now know, gravitated (via a stint at Novell) toward the Chairmanship of the very company that was by then most clearly demonstrating the accuracy of his 2003 vision.

So the answer to the $150BN vs $19.5BN market cap question above is that Google is still near the beginning of an Internet technology cycle that could last an entire generation, while Sun stands at the end of an i-Technology cycle that is already 24 years old.

But Bechtolsheim was once asked "So is the game over?" and I have never forgotten his reply: *"Only if no one changes the game."*

While the last few years may have been disappointing for people who thrive on accelerated progress in technology, the world is moving faster again. Lest I be accused of puffing air into Bubble 2.0, though – especially since this is the month when I typically poll so many minds for their *i*-Technology predictions – it would perhaps be as well if I were just to remind readers of technology visionary George Gilder's sobering words:

> "Amid the beckoning fantasies of futurism, the purpose of whatever comes next – like that of today's petapede – will be to serve the ultimate, and still the only general-purpose, petascale computer: the human brain."

Google figured the Web's first killer app. Web search now assists us *n* times a day in thinking, writing, and doing. And Google now helps us with communicating and social computing too. But if the network is the computer and the ultimate computer is the human brain, then maybe Java can help change the game by making the human brain the network? After all, where Eric Schmidt's Google goes, can Jonathan Schwartz's Sun – with its humongous R&D budget – be so very many steps behind?

Enjoy the technologically diverse predictions showcased in this issue. One thing alone is certain about the future: like it or loathe it, we're all headed there together!

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® XML Suite 2007 – The world's leading XML development tools.

Gear up for XML excellence

Take off with the Altova® XML Suite, and save ½ off the top tools for XML development.

Included with the Altova XML Suite 2007:

- Altova XMLSpy®, MapForce®, and StyleVision® Enterprise or Professional Editions
- Plus Altova SchemaAgent™, SemanticWorks™, and DiffDog® with Enterprise Edition
- Also get a FREE copy of Altova DatabaseSpy™ 2007 for a limited time*

The Altova XML Suite 2007 delivers the latest releases of world's leading XML development tools all in an unrivaled deal. It contains Altova XMLSpy, the industry standard XML development environment; MapForce, the premier data integration and Web services implementation tool; and StyleVision, the ultimate visual stylesheet designer. What's more, the Enterprise Edition also includes XML Schema management, Semantic Web, and XML-aware differencing tools. Save a bundle!

Download the Altova XML Suite today: www.altova.com

*__Special offer:__ Now until Dec. 24, 2006, purchase or upgrade to the Altova XML Suite and get the NEW Altova DatabaseSpy database query and design tool for FREE!

Conditions apply, see Web site for details.

DECEMBER 2006 VOLUME:11 ISSUE:12

JDJ contents

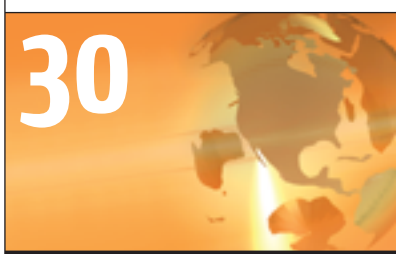
JDJ Cover Story

Concurrent Queries

A pattern for improving database query performance

by Andy Pardue 18

Features



30

The Development Power of Open Source AJAX Tooling
by Kevin Sawicki

44

j-Interop: An Open Source Library for COM Interoperability Without JNI
by Vikram Roopchand

52

Where's *i*-Technology Headed in 2007?
by Jeremy Geelan

JDJ (ISSN#1087-6944) is published monthly (12 times a year) for $69.99 by SYS-CON Publications, Inc., 577 Chestnut Ridge Road, Woodcliff Lake, NJ 07677. Periodicals postage rates are paid at Woodcliff Lake, NJ 07677 and additional mailing offices. Postmaster: Send address changes to: JDJ, SYS-CON Publications, Inc., 577 Chestnut Ridge Road, Woodcliff Lake, NJ 07677.

Nigel Cheshire
Guest Editor

How Good Is Good Enough?

Code quality and the point of diminishing returns

Intellectually everyone understands that improving code quality is a good thing. After all, we know *bad* quality when we see it. (Anyone old enough can cast his or her mind back to the late '80s and Microsoft Word for Windows 1.0.) But we also know that there comes a point where there's a diminishing return on our investment in code quality. How much work would you put into tracking down a bug that's only ever been reported once by a user running on OS/2 Warp?

The problem with code quality initiatives is that we really don't know how much effort to put into them. We have never truly answered the question: how much quality is enough?

Why Code Quality is Important

The Standish Group famously reports on the software industry's inability to deliver successful projects on a regular basis. In 2004, it reported that just 29% of software projects were considered a "success."

Projects fail for a number of reasons, and as far as the Standish survey is concerned, that means either the projects were completely canceled, or that they were "challenged" because they significantly overran time and/or cost.

There are many reasons that projects are canceled or overrun, and almost certainly the most common is that the software just didn't do what it was supposed to – it didn't conform to requirements. That's a hard problem to fix – and it's an even harder one to measure.

Code quality, on the other hand, which is probably the second largest contributor to project failure, is much easier to measure, and therefore easier to manage. Yet, many development teams waste time on rework cycles fixing bugs that crop up in QA or production that could have been caught and fixed upstream, with relatively little effort.

A big part of the cause of that problem is that during the coding phase, oftentimes the only measurement we have available to track progress is whether we hit the deadline or not. Even if we monitor defect rates (it may shock you to learn that not all development teams do), we just don't have any meaningful metrics by which to manage the process until it's too late.

Agile methods help. Test-driven development, where unit tests are written with, or even before the code, and continuous integration and test cycles flush out many bugs in development before they get to QA.

But fundamentally, managing only to deadlines causes an accumulation of "technical debt" – a term coined by Ward Cunningham to describe the effect of all those decisions to use a quick and dirty approach to solving a problem, with the idea that you'd come back and fix it later. Just like financial debt, technical debt isn't necessarily a bad thing, especially when an important deadline is looming. But like financial debt, if you have too much of it, it can get out of control and becomes very hard to pay back.

How Much Quality is "Enough"?

Quality is a tradeoff between cost and risk. If you ask most (internal or external) customers what level of quality they'd like to see in their software applications, they'll likely tell you "perfect" – i.e., zero defects. Of course, that's something of a trick question, because you didn't tell them how much "perfect" would cost.

–continued on page 10

Nigel Cheshire is CEO of Enerjy Software, a division of Teamstudio Inc. He oversees product strategy and has been driving the company's growth since he founded it in 1996. Prior to founding Teamstudio, Inc., Nigel was co-founder and principal of Ives & Company, a CRM solutions consultancy. He holds a Bachelor of Science degree in computer science from the University of Teesside, England.

nigel_cheshire@enerjy.com

SYS-CON MEDIA

President and CEO:
Fuat Kircaali *fuat@sys-con.com*

President and COO:
Carmen Gonzalez *carmen@sys-con.com*

Senior Vice President, Editorial and Events:
Jeremy Geelan *jeremy@sys-con.com*

Advertising

Vice President, Sales and Marketing:
Miles Silverman *miles@sys-con.com*
Robyn Forma *robyn@sys-con.com*

Advertising Sales Manager:
Megan Mussa *megan@sys-con.com*

Associate Sales Manager:
Kerry Mealia *kerry@sys-con.com*
Lauren Orsi *lauren@sys-con.com*

Editorial

Executive Editor:
Nancy Valentine *nancy@sys-con.com*

Production

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*

Web Services

Information Systems Consultant:
Robert Diamond *robert@sys-con.com*

Web Designers:
Stephen Kilmurray *stephen@sys-con.com*
Richard Walter *richard@sys-con.com*

Accounting

Financial Analyst:
Joan LaRose *joan@sys-con.com*

Accounts Payable:
Betty White *betty@sys-con.com*

Accounts Receivable:
Gina DeTemple *gina@sys-con.com*

Customer Relations

Circulation Service Coordinator:
Edna Earle Russell *edna@sys-con.com*

Speed. Simplicity. Style.

download the FREE trial now!

WINDOWS® FORMS ASP.NET WPF JSF

grids scheduling charting toolbars navigation menus listbars trees tabs explorer bars editors

> "Only when you have sufficient data to analyze can you start to make some decisions on how far to go with your code quality initiatives"

–*continued from page 6*

After a 1996 Fast Company article on the Lockheed Martin group that builds and maintains software for the space shuttle program, that software is often cited as the most expensive code on the planet, line for line. I'm not sure anyone really knows the cost per line of the space shuttle software (it's been estimated at $1,000 per line), but we do know that (as of 1996) it took roughly 260 developers to maintain 420,000 lines of code, which comes out at about 1,600 lines per person. That's expensive – but the approach seems to work: according to the article, the previous three versions of the software had only a single defect detected per release.

Of course, a typical business application isn't controlling a space shuttle (which, even in 1996 dollars, cost $4 billion a pop, not to mention the lives of a half-dozen astronauts being at stake), and many development organizations struggle with the cost/risk equation. How much quality is enough for your organization? For your application? How can we even start to answer that question?

Ideally, we should be able to come up with a mathematical solution. In probability terms, the expected cost of failure would be expressed as the cost of something bad happening multiplied by the probability that the bad thing will happen. So, as you spend more on increasing the quality of the software, the probability of failure comes down, reducing the expected cost of failure.

This approach becomes problematic in practice, because it's hard to measure the incremental cost of increasing quality and the probability of failure. But, unless we (as an industry) start to measure these things, we'll remain in the Dark Ages.

What Gets Measured Gets Managed!

So, what to measure? A good place to start is the cost of fixing a defect. Once you know the cost of fixing a defect, and you start to measure defect rates, you can assess the return on the investment spent on quality measures (which should be driving defect rates down).

The cost to fix a defect depends on many factors. Conventional wisdom states that the earlier in the software lifecycle a defect is found and fixed, the lower the cost. Of course that's true, although depending on the application, it probably costs less to fix an application in production now than it did even a few years ago. (How many times have you opened up an application only to have it ask if you want to automatically download and install the latest and greatest version?)

There are other quality-related metrics that can help get a handle on how well code is being written. Most development organizations have a set of coding standards. How many organizations measure how well the standards are being applied? Coding standards don't just tell you where to put your curly braces. If applied consistently, they can have a real, measurable impact on code quality and maintainability.

There are a number of static code analyzers available that help development teams implement their coding standards. And static code analysis can go further than just checking adherence to standards – it can find bugs that would otherwise slip through the compiler undetected (consider the old chestnut of incorrect use of the "=" operator in a conditional statement).

Unit test activity is another early indicator of software quality. Although it's not perfect, code coverage (the percentage of executable lines of code exercised by your unit test suite) is a good measure of the effectiveness of your unit tests. You have to keep in mind of course that code coverage isn't the same as path coverage (in other words, 100% code coverage doesn't mean that your code is bug free!). But it's still fair to say that, up to a point at least, the higher your code coverage percentage, the better chance you have of squeezing those bugs out.

This is a great example of a place to look closely at return on investment. You should set a realistic goal for code coverage. A development team could spend many hours trying to get the coverage rate from 98% to 100% – with very little return in terms of the number of defects trapped in development. Many of the better-known open source projects show coverage rates in the 50% to 70% range, and I know for a fact that many commercial projects sit at about half that number.

The important point here, to roll out a well-worn adage, is that you can't manage what you don't measure. If you're not measuring defect rates at different stages of the lifecycle, start now. Take a look at some of the tools available for static analysis and code coverage and, more importantly, start to track those quality metrics over time. Only when you have sufficient data to analyze can you start to make some decisions on how far to go with your code quality initiatives, and finally be in a position to answer the question: how good is good enough?

top **MISCONCEPTIONS** that drive

Meet the most misunderstood developer team in the world.

our Crystal Reports dev team crazy

Crystal Reports® is too expensive. Actually, the developer edition is just $595[1] USD (or upgrade for only $315[1]). Complimentary Crystal Assist support[2] provided with purchase.

Crystal Reports doesn't include a free runtime license. Not true, the developer edition includes a free runtime license[3] for each component engine.

Getting reports on the web is complex. False, the developer edition includes crystalreports.com[4] and Crystal Reports Server[5] to speed and simplify web reporting deployments.

Crystal Reports only works in Windows®. Not quite, whether you need to create or deploy reports on Windows, Linux or Unix, we have a Crystal Reports technology for you.

Find out more at: www.businessobjects.com/devxi/misunderstood

Business Objects™

1 Suggested retail price. 2 Complimentary access to support engineers and self-help. 3 Includes an unlimited runtime license for internal use of .NET, Java, and COM engines. 4 Includes ten named user licenses. 5 Includes five named user licenses.

Is Your Mobile Safe?

How to avoid the 'blues'

by Kanchan Waikar

We don't forget to scan our PC for viruses and worms but we conveniently forget to download a virus checker for our mobile. Most of us are still under the impression that mobiles are completely secure, which isn't true. There are a number of threats that can crash your mobile handset. Since few people have suffered from such harmful programs, mobile attacks haven't gotten much publicity. In personal computers, viruses attack through removable drives and Internet attachments whereas M-viruses and worms attack through SMS, downloadable application files, Bluetooth, etc.

We have very little knowledge about such mobile attacks. There are many mobile viruses that can even crash your mobile handset. Having a Bluetooth facility in a mobile device makes it even more vulnerable. Bluejacking, bluesnarfing, and backdoor attacks are some Bluetooth attacks. The attacker can access your contact data, personal images, and SMS inbox after making a Bluetooth connection with your mobile.

There are two "free" ways of sharing information that are available with most of the mobile handsets. These are infrared and Bluetooth. Bluetooth offers better coverage compared to infrared. Hence there's greater possibility of attack through Bluetooth than infrared. Let's look at some of the popular Bluetooth attacks.

Bluetooth Attacks...

The simplest one is bluejacking. It is the process of sending anonymous messages using Bluetooth technology. In bluejacking we can send a message, a video file, or an audio file. Bluejacking can't harm the destination mobile. When bluejacking is done to advertise or spam the victim's inbox then it's called bluespamming. In bluespamming, the interloper sends spam messages to all mobiles in its Bluetooth range.

So much for the "not so harmful" attacks. Now let's look at real Bluetooth attacks. The most popular Bluetooth threat is a bluesnarfing attack. Bluesnarfing is dangerous because it can steal your address book, your personal data, and do *n* numbers of such harmful activities. A bluesnarfing attack can take place from any well-equipped Bluetooth-enabled device. Bluesnarfing can update the most sensitive data and doesn't leave any traces behind. Using bluesnarfing, the attacker can update your address book. A bluesnarfing attack can also set call forwarding in train and cost you money. Then there's bluebugging – the attacker makes calls from the victim's mobile handset remotely.

In Bluetooth, the packet-size value is set for different devices. There's an attack that uses this to try to crash the victim's handset. If a Bluetooth device gets a packet of greater size than its allowed limit it crashes, hangs, or sometimes simply reboots. This attack is known as bluesmacking.

Viruses, Worms & Trojans...

Other than Bluetooth attacks, there are other threats like worms, viruses, and Trojan horses.

Let's start with the first mobile Trojan "Mosquit.a." It doesn't hurt your handset but it costs you money. It's basically a mobile game that sends numerous SMS messages to different numbers when you're busy playing the game. Then there's a *Commwarrior* virus that spreads either through Bluetooth or MMS and attacks 60 series handsets. There's a *skulls* Trojan that disables all applications and replaces all application icons with the image of a skull. It's basically a SIS application that replaces all application software with wrong versions disabling basic functionality. And there's is a Trojan called "SymbOS.Locknut" that can crash a victim's handset. If you get a call that displays the name "ACE" and if you pick the call up, it can erase your IMEI number and make your phone useless. Then there's a "Drever-C" Trojan that poses as a security update and corrupts the boot loader. So be careful downloading applications from the Internet.

How Can I Protect My Mobile?

- Don't open untrustworthy applications
- Don't pair your device with unknown devices
- When entering a crowded zone, make sure your Bluetooth is switched off
- Keep your mobile anti-virus updated

In case viruses, Trojans or worms are detected, anti-virus companies have patches and updates. To protect mobile handsets you have to keep your mobile anti-virus updated. If you find that your cell's been attacked by some virus or Trojan don't reboot it. Some Trojans affect the boot loader/boot data and rebooting will make it hard to repair.

Conclusion

After reading about the different kinds of attack that can take place, you might be worried about your cell's security. But we can still keep our mobiles safe by having the appropriate anti-virus loaded on them and keeping our Bluetooth on only when needed. An option in our cell phones lets us maintain a "Hidden/Invisible" connection that refuses new Bluetooth connections, but some devices still remain vulnerable. So keeping Bluetooth on only when required is the only option that remains. We can also refuse to accept connection requests from unknown devices.

Well-known PC anti-virus companies like Symantec and McAfee provide anti-virus protection for mobiles.

References

- http://news.zdnet.co.uk/communications/wireless/ 0,39020348,39145881,00.htm
- An Ethical guide to hacking mobile phone by Ankit Fadia, Macmillan publications
- http://www.antivirusprogram.se/virusinfo/SymbOS.Locknut_3172.html
- http://www.viruslist.com/en/analysis?pubid=200119916
- http://hoaxbusters.ciac.org/HBMalCode.shtml

Dedications

I would like to dedicate this article to my best friends and my mom and thank my friend Ms. Sneha Abraham for her valuable suggestions.

Kanchan Waikar is a software professional working with a multinational IT company and much inclined to mobile programming.

waikar.kanchan@gmail.com

MAKE ANSWERS TO PERFORMANCE PROBLEMS COME TO YOU.

OPNET Panorama offers powerful analytics for rapid troubleshooting of complex Java EE applications. Panorama quickly identifies how application, web, and database servers are impacting end-to-end performance. With Panorama, you can pinpoint the source of a problem, so time and money aren't spent in the wrong places.

The world's most successful organizations rely on OPNET's advanced analytics for networks, servers, and applications.

OPNET Panorama
Real-Time Application Analytics

www.opnet.com/panorama

OPNET®
Making Networks and Applications Perform™

OPNET Technologies, Inc. 7255 Woodmont Avenue, Bethesda, Maryland 20814 phone: (240) 497-3000 • e-mail: info@opnet.com • NASDAQ: OPNT

The Challenges of Porting a Java ME Application to Multiple Devices

by Hayden Marchant

Coping with idiosyncrasies, the unexpected, and complexity

The familiar phrase attributed to Java applications of "write once, run everywhere" sadly does not apply to applications developed on Java ME.

While the Java ME standard ensures that runtime environments are consistent across devices, the many idiosyncrasies that exist outside the Java specification require that all ME applications be tailored to each device. The differences in behavior can range from unexpected exceptions when calling certain API calls to performance issues in certain operations on the device. These problems cause many difficulties for developers, who have to ensure that their applications will work well on all required devices and to an acceptable level.

Fixing a bug on one device can often cause a worse bug to emerge on another. In turn, adding simple features to an existing application can turn into an inflated task due to the unexpected behaviors that some devices take on when doing seemingly trivial operations. This makes testing Java ME applications that are being deployed on multiple devices a very complicated task that needs careful project planning to cope with the unexpected and manage the large volume of platforms on which to test.

So porting a Java ME application to multiple devices can be a daunting task. In this article, we'll discuss some of the common challenges encountered when developing a Java ME application designed to run on multiple devices and some approaches to overcoming them.

Hayden Marchant is a Software Engineer in the Information Integration Solutions group at IBM. He has 10 years of experience in software engineering. Hayden has a 1st class honors BA in mathematics from Cambridge University, England, and is a Sun Certified Programmer.

hayden@il.ibm.com

A View into the Java ME World

For a thorough understanding of the complexities of Java ME projects, let's walk through the lifetime of a Java ME application project.

First Steps

The typical Java ME application is generally developed against a standard Java ME device. Usually it will be a middle-of-the-range device, most probably selected following the suggestions of the target mobile operator.

One of the first decisions that the developer makes is selecting an IDE. A few IDEs support Java ME application development, NetBeans being the most mature. In the last year, both Eclipse and Intellij IDEA began to catch up and released versions of their IDE with more than sufficient support for Java ME development. The IDE for a Java ME project should be selected against the same criteria as other Java projects – basically, anything that helps the programmer with productivity, while ensuring quality code. Since the introduction of *ant* into the mainstream Java programming world, cute wizards/build utilities that certain IDEs offer are less of a buying point than they used to be.

After selecting the relevant Java ME emulator, the developer will tend to spend a large chunk of his development time testing the current status of the application on the Java ME emulator, and only towards the end will the application be tested on the actual device. Very often new bugs will arise when testing on the device, so deploying to the device should be done as early and as often as possible.

Unit Tests

As with all Java applications developed, it's imperative to write unit tests for the Java ME application. This becomes even more important when the number of devices needing to be supported increases, since unit testing will be one of the principal methods of weeding out different behavior in devices at an earlier stage in the development cycle.

The specific runtime environment in which the unit tests are executed becomes increasingly important as the number of supported device increases. For the first device it's generally acceptable to run the tests on the emulator most of the time, and every now and then to run the tests on the actual device.

Since most Java ME applications are heavily GUI-oriented, it's imperative to separate the layers of the application in a typical MVC pattern. This way it will be possible to unit test much of the Model and Controller part of the application. (Special attention should be paid to the memory cost of each additional class when the device has small JAR size restrictions. However, this is becoming less relevant as most devices on the market today have respectable JAR size capabilities.) If this separation isn't done, it will substantially reduce the benefit of the unit tests and you'll have to rely heavily on manual testing your application – something you want to avoid at all costs, since each device is, in essence, a different platform. So, the need to undertake manual testing should be reduced as much as possible in these areas by covering this functionality with unit tests.

J2MEUnit is the natural choice for a unit test framework. It's based on the popular JUnit framework and can be run as a J2MEMIDlet suite.

Porting to Other Devices

When the application is working at a reasonable quality level, demands for support on extra devices usually roll in. At the

stage you might suddenly be presented with a list of 20 devices you've never seen before, with your customer inevitably asking, "How long will it take to get it running on these?"

This is the real challenge in Java ME development – porting to multiple devices. The challenges are multi-faceted. In the world of the unknown, you have to understand the cost of porting the application to each device as early as possible, while always ensuring the high quality of the application!

Get To Know Your Device

To provide an accurate cost estimate for a device, it's to know your device. There are several bits of information that you have to know. Some of this information will be documented on the Web, and some you'll only be able to discover by running tests on the device.

Most device vendors have a Web site devoted to application developers. On these sites you can generally find useful information, whether it be in an official Developers Guide published by the vendor, or in a post on the site's forum. At one end of the spectrum, there are vendors that have sites with a large, active community of developers, as well as great documentation. At the other end, there are vendors whose developer sites are either non-existent or so poorly supported that they may as well be non-existent. Table 1 shows some of the larger device Web sites.

You should aim to find out as much as possible about the following aspects of the device from these external resources:

1. Which emulator to use
2. Developer Guides
3. I/O capabilities
4. UI specifications
5. API specifications
6. Supported JSRs
7. Memory limitations
8. Security/signing issues
9. Known issues

Depending on the functionality of the application, you'll typically need a variety of information for your device. Some of it will be formal definitions and capabilities that can be obtained from online resources.

War Stories

The following are typical examples of problems that can occur on devices. I can truly say that most devices have at least one war story, albeit not as severe as some of these examples.

Device Vendor	Site
Motorola	http://developer.motorola.com
Nokia	http://www.forum.nokia.com
Samsung	http://developer.samsungmobile.com
Sony Ericsson	http://developer.sonyericsson.com

Table 1

Requirement: Your application needs to read and write up to 500KB of data to local storage on the device. This capability is something that can usually be discovered on the Web, and can be extremely valuable in providing estimates for particular devices.

Limitation A: According to the official Web site, Device X can only hold 256KB in its RMS (Record Management System, which is an API for persistent storage for Java ME devices). To support this device, an expensive change in the product's architecture might be required in which data is saved to a remote server. However, before believing the documentation, you run a quick check to confirm this limitation.

Limitation B: According to the Web site, Device X can hold 500KB. However, when running the application, it works satisfactorily until the amount of data saved on the device reaches 150KB after which the application crashes. This is an example of an undocumented restriction. To support this device, research will be required and possibly a change in the product architecture.

Requirement: Your application displays a set of graphic images that fit neatly on a 208*208 screen.

Limitation A: Device X's screen dimensions are only 200*200, which means that the images won't fit on the device. To support it, a new set of images will have to be created that will fit on this smaller device. The application's image layout algorithm will also have to be enhanced (if it ever existed).

Limitation B: Device Y's screen dimensions are big enough, but when the application runs, besides it taking nearly 10 seconds to paint the images to the screen, they appear very strange. After a lot of painstaking research and debugging, the device is discovered to be notoriously slow at rendering images to the screen and doesn't support the exact format of the current images. After generating very low-resolution images and changing the device's image display algorithm, you manage to get the time down to five seconds and the images look reasonable.

Requirement: Your application gets incoming messages through SMS messages received on a specific port.

Limitation A: When Device X gets an SMS message on this port, the application gets the message correctly. Device X is considered a new state-of-the-art device and has excellent performance and API capabilities. Shortly before releasing the application on this device, it's discovered that if two or more SMS messages are sent to the device within 10 seconds, the application crashes and the device has to be rebooted! This turns out to be a bug in the device's operating system and the only workaround is to develop an alternative communication channel using TCP/IP. This means developing a server-side component to manage this new mode of messaging plus a different cost-model for the mobile provider to handle this application's messaging.

Figure 1 I/O testing menu

Figure 2 SMS receive testing

Figure 3 RMS capacity testing

Figure 4 Screen properties testing

Figure 5 System abilities testing

Limitation B: Installing the application frequently fails on Device Y. After days of painstaking research, it's discovered that if the port number that the SMS messages come in on is changed to a different range, the installation completes and the application works fine. The mobile operator is perturbed at the news of the change in port number since it doesn't usually support messaging over those ports, and following this, there's a substantial amount of annoying politics to go through to get the change approved.

Reducing Risk

To decrease the element of surprise in porting the application to a device, it's advantageous to discover as many of the problems as early as possible. Many of these issues can be discovered both in the online resources and by running a thorough set of unit tests on your suite.

However, to add a greater level of protection, another level of testing framework can be used that will expose many of the problems mentioned above, which would otherwise be discovered much later in the development cycle. This testing framework is, in essence, a tailor-made application that has several sections to it. Each section should cover a large area of functionality, like image rendering, key

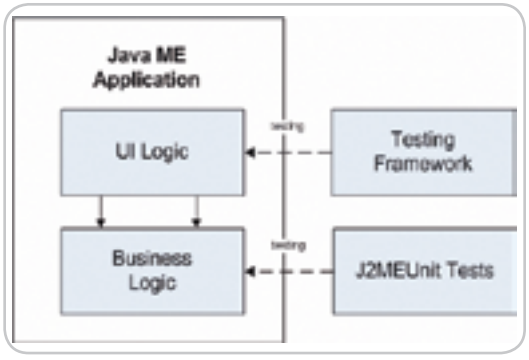


Figure 6 Testing architecture

mappings, menu behaviors, internationalization, GUI, I/O, or performance, with menu items that test each of these areas (see Figures 1–5). These tests cover areas of the application on a higher level than regular unit tests since there are some areas where automatic unit testing can't be applied such as performance issues and GUI errors.

The testing framework can typically contain several sections from which manual testing of application behavior can be carried out. A detailed test plan should accompany the test framework so the tester understands the success or failure of any particular test. This test package, consisting of the test framework together with the test plan, is an excellent tool to have when initially analyzing a device – it can make all the difference in arriving at accurate time estimates.

The testing framework should be updated as new functionality is added to the application. Just as important, whenever serious problems are discovered in a device, new tests should be added to detect the same problems in future devices. Only this way is it possible to add support for a large number of devices in a scalable manner.

With a good testing framework and unit test suite, the development team should feel significantly more comfortable in committing to new devices (see Figures 6).

However, for any Java ME application porting project to succeed, the customer must always keep the priority of each device in the back of its mind, since, for example, it would be a bad idea for a mobile operator to commit to two months of redesigning a device that only covered 0.3% of its mobile device market, where the other 99.7% work well. On the other hand, if that device had 25% of the market, it might be the correct decision to take.

Device Vendor	Site
ICON_SIZE	16
FONT_SIZE	7
COMM_PROTOCOL	XYZ
SCREEN_WIDTH	128
SCREEN_HEIGHT	128
KEY_MAPPINGS	NOKIA_STANDARD
ENABLE_SOFT_MENU_DOUBLE_CLICK	False
RMS_WRITE_INTERVAL_MS	200

Table 2

Keep the Code Clean

In the same way that a thorough level of testing is required, it's also very important to keep the code base clean and easy to maintain. Without this, the code base can quickly become very messy in direct proportion to the number of devices supported. When each device has slightly different behaviors, size differences, etc., the code will slowly begin to get scattered with # if preprocessor directives.

However, it's important to keep the code clean of directives that refer to device names. Instead, the directives should refer to semantic capabilities of the application. For example, Device Y requires a slightly different address format in the communications module than all other devices due to its using a different communications protocol. Instead of having code like:

```
#if DEVICE=="Y"
address = formatDifferentAddress(address);
#endif

sendMessage(address,portNumber,content);
```

the code should read:

```
#if COMM_PROTOCOL=="XYZ"
address = formatDifferentAddress(address);
#endif

sendMessage(address,portNumber,content);
```

The different behaviors should be extracted out as capabilities, and devices are attributed with these capabilities. For each device there will be a configuration file that contains details of the capabilities that the device requires, which, at build time, is translated into the correct preprocessor definitions. Table 1 illustrates an example device capability file. Observe that some of the capabilities are feature-based in that each device is expected to have a different value, like screen settings, and other capabilities are the result of bugs in the system. For example, a bug in the RMS writing module on a certain device could have resulted in special code that writes data at set intervals, which results in the RMS_WRITE_INTERVAL_MS capability.

Since ant is the most common way to build Java applications, it's recommended that this file be a properties file (see Table 2).

As with most tasks in the Java world, there's a tool that can help you manage device capabilities. J2MEPolish is an advanced open source build tool and GUI framework for Java ME applications and currently has a database of nearly 400 devices with a lot of predefined capabilities. Another recently released open source library, the J2ME Device DB project, boasts capabilities similar to J2MEPolish and is worth evaluating. There are also several commercial packages available that address these problems that can be found on the Web.

Flexible Builds

Having a Java ME application that's supported on a large number of devices means that you have to make a large number of builds when releasing versions for testing or QA. Manually building devices might be acceptable for a few devices, but when the number increases, you'll have to have a way of batch building applications for all devices or for a particular subset of devices. Again, J2MEPolish has a good ant-based build framework sufficient for most needs.

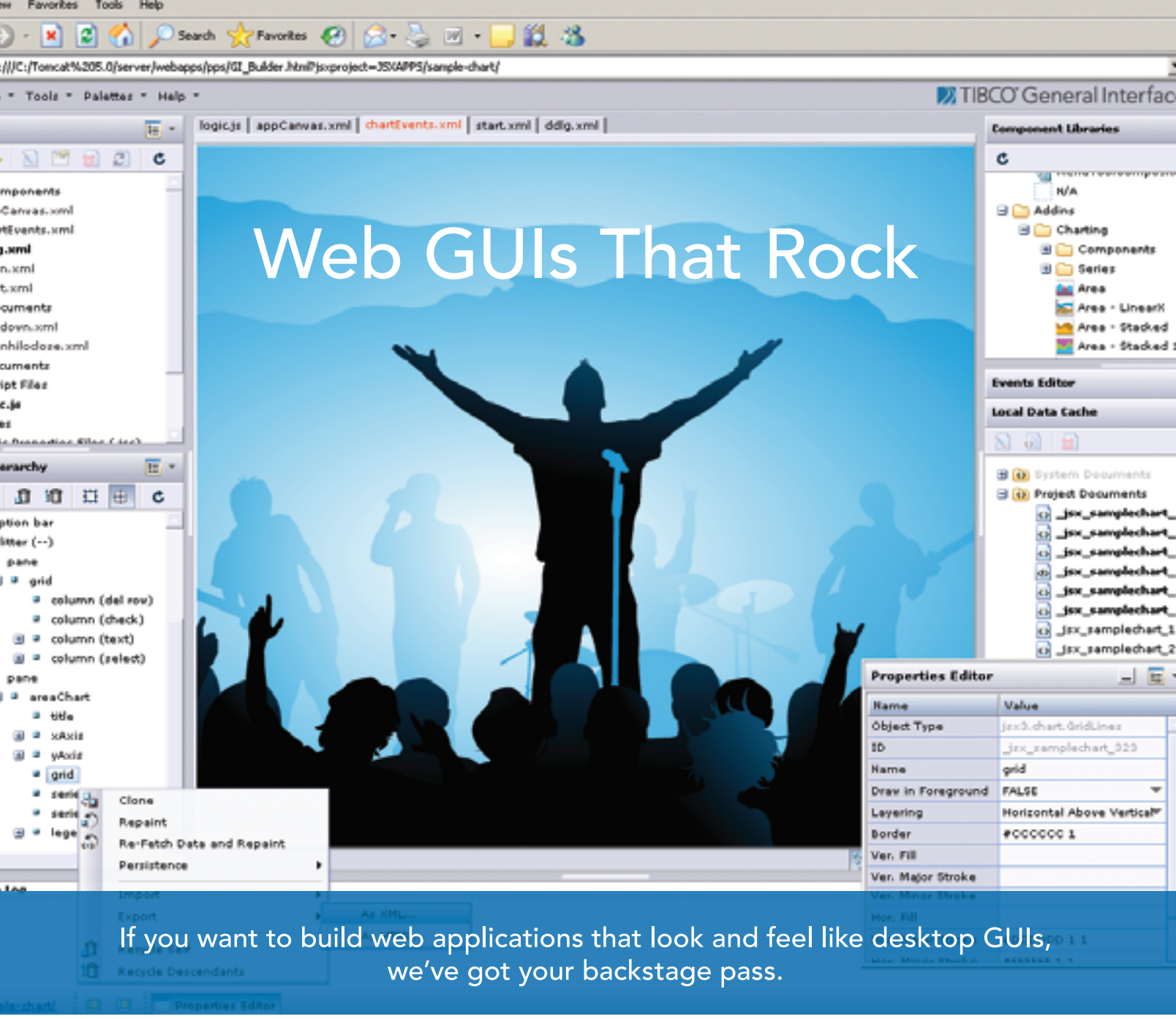
However, when each customer needs multiple languages, and possibly different feature sets, the build process will get more complicated. For each device there might be different language options combined with several different feature sets that have to be built. The difficulty of managing which configurations to build for which customer becomes increasingly relevant and a solid, flexible build is mandatory to carry the project through this complexity

Conclusion

Porting Java ME applications can be scary, but with a lot of preparation and research, Java ME applications can be successfully ported to multiple devices. Fears about being swamped by the unmanageable number of problems that are encountered in different devices can be silenced. The ME project, which could so easily have ended in failure, can now be tackled with the same confidence that one has undertaking a regular Java project.

References

- J2MEUnit – http://j2meunit.sourceforge.net
- JUnit – http://www.junit.org
- J2MEPolish – http://www.j2mepolish.org
- J2ME Device DB project – http://j2me-device-db.sourceforge.net



Why are professional developers choosing TIBCO General Interface?
With more components and tools, TIBCO General Interface™ enables you to create AJAX applications that look and feel like desktop GUIs with astounding speed. No wonder it's the #1 independently rated AJAX Rich Internet Applications toolkit.

SOA and AJAX Rich Internet Applications are changing the way software is developed and deployed. TIBCO helps you make the shift from 3-tier to SOA-based computing to deliver astoundingly rich and agile solutions fast.

TIBCO®
The Power of Now®

Are *you* ready to rock? Learn more at http://developer.tibco.com/.

Concurrent Queries

A pattern for improving database query performance

by Andy Pardue

Does this sound familiar? You have a domain object, perhaps for reporting purposes, that's built from a ton of JDBC queries and it takes too long to load. Nothing else happens until this object is built, so it's become a bottleneck. Even worse, each of the queries is actually well tuned, so there isn't much to gain from modifying the queries themselves – there are just too many of them. You don't want to change (or can't change) your data model, so what can be done to alleviate this problem short of a major redesign? There are several options like caching, lazy loading, resource pooling. Another worthy option would be to implement a variation of the concurrent query pattern.

Concurrent queries are fairly simple to implement and even simpler to describe. Rather than serializing a set of queries, one after the other, waiting for one to complete before the next begins, one would actually use threads to run sets of independent queries simultaneously. Now, using threading for database I/O might sound daunting or ill-advised, but the Java threads package is one of the best, if not the best, I've had the pleasure of working with. Plus, the new concurrency utilities supplied with Java 5 make using threading for database I/O much more feasible. The UML-ish diagram in Figure 1 attempts to provide a pictorial representation of what I hope to explain in the following paragraphs.

Suppose a domain object is built from 200 queries that can each run from execution to processing results in 100 milliseconds on average. Now 100ms isn't too bad for a query, but stacked end to end, the result would be up to 20 seconds to build an object. Ouch. Now, suppose you could run up to five of these queries concurrently at any one time. In an example that will be described later, I was able to take a similar scenario and increase performance from 20+ seconds to build an object to 5+ seconds. First, though, I'll describe a simpler problem scenario and then implement a solution using concurrent queries, making use of JDBC, connection pooling, and Java 5 thread pools in an effort to demonstrate the type of performance improvements this pattern might render. Later on, I'll cover another, more complicated implementation of an object that's built from several actual database queries. Note: Full source code for these sample implementations is available on sourceforge.net.

Andy Pardue is a senior software developer who has specialized in the medical software industry for over 15 years, 11 years as a telecommuter from his home office in Mesquite, Texas.

andypardue@gmail.com

A Serialized Baseline Sample Application

First, let's look at the usual case for building objects from database queries. For the following example, a user-defined sleep function was created in a Postgres database (using Postgres magic that I found on the Internet) that could be called as select sleep(N) where N is the number of seconds to sleep. After sleeping for the seconds indicated in the argument the number of seconds slept is returned back as a result. We'll do this with a class called SleepyObject in Listing 1.

This is a fairly standard JDBC query class that selects the sleep(N) function for the number of seconds desired and processes the ResultSet, which simply contains an integer indicating the number of seconds requested to sleep. So, if you "select sleep(1)," the result of that query will be 1. Effectively, the sleep function mimics a query that takes N seconds to complete. In the serialized example (Example1.java), an array of five SleepyObjects is created. Upon creation, each SleepyObject selects the sleep(N) function via JDBC and processes the result. This example creates an array of SleepyObjects then iterates over that array, printing out the return value from the call to the sleep function. Then, the number of seconds it took to execute the entire exercise is printed. Since this sample creates JDBC objects in the usual way, the second SleepyObject isn't created until the first object is fully created (e.g., finished with its sleep(N) query), and so on. So, in this example, since each SleepyObject sleeps 2, 1, 2, 2, and 1 seconds, respectively, the entire application must take at least eight seconds plus overhead to run, as indicated in the output below. This is Example1.java - a serialized example.

```
package net.sourceforge.concurrentQuery.article.serialized;

import java.sql.SQLException;

public class Example1 {

    public Example1() {}
    public static void main(String[] args) throws SQLException {

        long start = System.currentTimeMillis();

        SleepyObject[] sleepyObjects = {     new SleepyObject(2),
                                             new SleepyObject(1),
                                             new SleepyObject(2),
                                             new SleepyObject(2),
                                             new SleepyObject(1)
                                           };
        int i = 1;
        for (SleepyObject sleepyObject : sleepyObjects) {
                System.out.println("SleepyObject " + i++ + "
```

```
returned "
                                    + sleepyObject.getValue());
        }
        long end = System.currentTimeMillis();
        System.out.println("took: " + new Double(end - start) /
1000 + " seconds");


    }
}
```

The following is output from Example1.java.

```
run-example1:
     [java] query is: select sleep(2)
     [java] query is: select sleep(1)
     [java] query is: select sleep(2)
     [java] query is: select sleep(2)
     [java] query is: select sleep(1)
     [java] SleepyObject 1 returned 2
     [java] SleepyObject 2 returned 1
     [java] SleepyObject 3 returned 2
     [java] SleepyObject 4 returned 2
     [java] SleepyObject 5 returned 1
     [java] took: 8.54 seconds
```

An Implementation Using Concurrent Queries

Next, we'll build on the same example, but this time we'll use a class called ConcurrentSleepyObject to replace SleepyObject. The ConcurrentSleepyObject will use a singleton implementation of the concurrent query pattern to invoke queries and reap the results as seen in Example2.java below.

```
package net.sourceforge.concurrentQuery.article.concurrent;

import java.sql.SQLException;

public class Example2 {

    public Example2() {}
    public static void main(String[] args) throws SQLException {

        long start = System.currentTimeMillis();

        ConcurrentSleepyObject[] concurrentSleepyObjects = {
                                    new ConcurrentSleepyObject(2),
                                    new ConcurrentSleepyObject(1),
                                    new ConcurrentSleepyObject(2),
                                    new ConcurrentSleepyObject(2),
                                    new ConcurrentSleepyObject(1)
                                            };
        int i = 1;
        for (ConcurrentSleepyObject concurrentSleepyObject :
           concurrentSleepyObjects) {
                System.out.println("ConcurrentSleepyObject " + i++
                   + " returned " + concurrentSleepyObject.getValue());
        }
        long end = System.currentTimeMillis();
        System.out.println("took: " + new Double(end - start) / 1000 +
           " seconds");

    }
}
```

Both Example1 and Example2 build an array containing five objects, each invokes a database sleep for the same amount of time. However, in this second example, by using an implementation of the concurrent query pattern, the same JDBC calls can be executed in less than half the time (2.86 seconds versus 8.54). This is because we have five queries running at once rather than one at a time. In this example, each of the five queries is run and resolved within its own thread. So, rather than the total execution time being the sum of all queries plus overhead, as in the first example, the total execution time is roughly the amount of time it takes for the longest query to run plus overhead. Here is the output from Example2.java shown below.

```
run-example2:
     [java] query is: select sleep(2)
     [java] query is: select sleep(1)
     [java] query is: select sleep(2)
     [java] query is: select sleep(2)
     [java] query is: select sleep(1)
     [java] ConcurrentSleepyObject 1 returned 2
     [java] ConcurrentSleepyObject 2 returned 1
     [java] ConcurrentSleepyObject 3 returned 2
     [java] ConcurrentSleepyObject 4 returned 2
     [java] ConcurrentSleepyObject 5 returned 1
     [java] took: 2.86 seconds
```

To accomplish this, a class called ConcurrentQueryThreadImpl.java was created as a singleton class that encapsulates:

1. The number of active queries that can run at once (five for purposes of this example).
2. A ConcurrentHashMap to hold a list of running query threads and a reference to the domain object interface to be used to reap the results of the query. ConcurrentHashMap is a thread-safe HashMap available in the java.util.concurrent package.
3. A second ConcurrentHashMap to hold a list of queries and domain object interfaces of queries that couldn't be immediately submitted because the maximum number of threads (five) was already running.
4. The needed JDBC code to execute the queries.

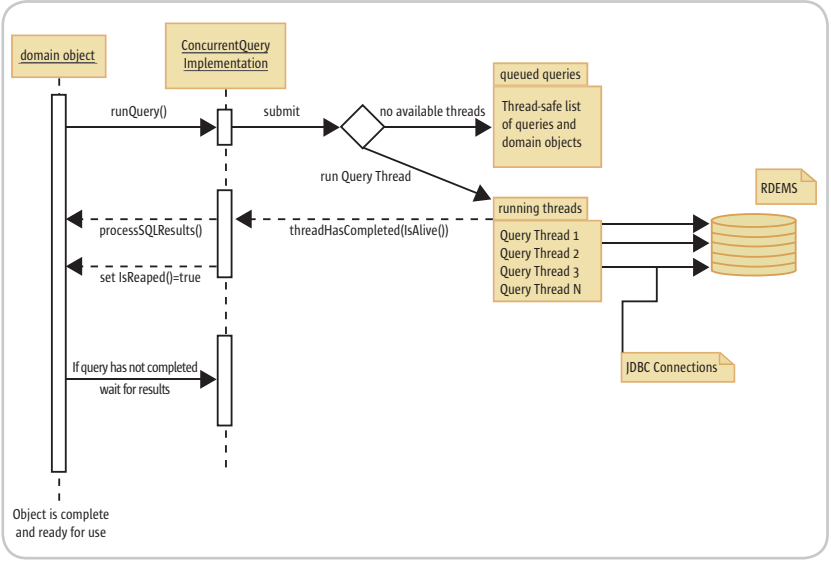


Figure 1

The code fragment in Listing 2 shows the initialization of the ConcurrentQueryThreadImpl class.

The interface CanResolveAConcurrentQuery, referenced in the ConcurrentQueryThreadImpl class, is used in the ConcurrentHashMaps and simply defines two methods that must be implemented by a participant in a concurrent query (e.g., ConcurrentSleepyObject), one to process the SQL results and another (isReaped())method that the ConcurrentQuery implementation can use to indicate to the object that it has processed its SQL results and is ready to go. Below is CanResolveAConcurrentQuery.java.

```
package net.sourceforge.concurrentQuery.article.concurrent;

import java.sql.ResultSet;
import java.sql.SQLException;

public interface CanResolveAConcurrentQuery {
    boolean processResultSet(ResultSet rs) throws SQLException;
    void setReaped(boolean isReaped);
}
```

By implementing this interface, the ConcurrentSleepyObject can participate in concurrent queries. Notice that in the getValue() method the query object needs to make sure that it has been "reaped" (e.g., it either processed its results or threw an SQLException) and if not, the object must call the waitForAllQueriesToComplete() method of the ConcurrentQueryThreadImpl singleton, which submits and processes all outstanding queries. See getValue() method of ConcurrentSleepyObject shown in the following.

```
public class ConcurrentSleepyObject implements CanResolveA-
ConcurrentQuery {

...
    public int getValue() throws SQLException {
        if (!reaped) {
            ConcurrentQueryThreadImpl.getInstance().waitForQu-
eriesToComplete();
        }
        return value;
    }
...
```

This is done so that the object can be sure that its results have been processed before it can be used. The waitForAllQueriesToComplete() method won't return until all running and queued queries are finished. This way, our object can be sure that its results have been processed before continuing. A better option, though, would be to assign a token, or cookie, to each participant in a concurrent query that can be used by the object to ensure that results are ready. This way, if the results aren't ready yet, the object won't have to wait for all the other queries to finish, but could be notified when its query has completed, perhaps moving it to the front of the queue, if necessary. To keep things simple, I opted for the main –strain-and-brute-force approach of waiting for all queries to finish. The complete source for the ConcurrentSleepyObject class is in Listing 3.

Details of the ConcurrentQuery Implementation

As mentioned, this implementation uses two lists to manage queries. The first list is the threads that are currently running SQL queries and can't surpass the configured value for the application. Since each thread corresponds to a JDBC connection, you want to be careful not to set this value too high. I've used five for this example. The second list is the queued queries that haven't been able to run because the running thread list was full when the query was submitted via the runQuery() method. See Listing 4.

In the ConcurrentQueryThreadImpl class, the runQuery() method first checks to see if any previously submitted query threads have finished and need to be reaped. This is important because the list of running threads is constrained so that too many queries can't run at once and overload the database server. So we want to get these threads processed and off the list first to make room for more query threads to be invoked. Once a query thread has been reaped then there's room on the list for another query thread. If there's room on the running threads list and there are queued queries waiting to be submitted (e.g., queries that previously had to wait because the running thread list was full) then they get submitted first before the query being passed to the runQuery() method. The query being passed in would then have to go onto the end of the list. Otherwise, if there's room on the running threads list and no queued queries, the caller's query will be immediately submitted.

The ConcurrentQueryThreadImpl class contains a private QueryThread class that extends Thread. This class starts a new thread, runs the SQL query, and holds onto the results (or an SQLException, if one occurred) until the ConcurrentQueryThreadImpl processes the results and removes the thread from the list. See Listing 5.

Once the ConcurrentQueryThreadImpl notices that the QueryThread is finished, it calls the processResults() method of the CanResolveAConcurrentQuery interface reference that the domain object has implemented, marks the processed object as reaped via the same interface, and removes the QueryThread from the list of running threads. Besides the getInstance() method that gives visibility into the singleton class, the public user interface for this class simply consists of the runQuery() and waitForAllQueriesToComplete() methods.

A Variation Using Thread Pools

In situations where concurrent queries can be used extensively, there might be some uneasiness about starting a new thread for each query and having it exit when that query is completed. In such cases, I'd recommend using a callable thread pool available in the java.util.concurrent package. Threads of this type would have an advantage over normal threads in that a) they can be pooled, b) they can throw an exception, and c) they can return a result. As an exercise, I've implemented a callable thread pool version of the QueryThread class that the ConcurrentQueryThreadImpl class can use to run queries. This class, a private class named QueryThreadPool, implements the Callable interface, instantiates a thread pool the size of the constraint of the maximum number of queries we want to have running

BACKBASE

AJAX for Java

Backbase offers a comprehensive AJAX Development Framework for building Rich Internet Applications that have the same richness and productivity as desktop applications.

The Backbase AJAX Java Edition:

- is based on JavaServer Faces (JSF)
- runs in all major Application Servers
- supports development, debugging and deployment in Eclipse
- embraces web standards (HTML, CSS, XML, XSLT)

Download a 30-day Trial at www.backbase.com/jsf

i: www.backbase.com **t:** (866) 800-8996 **e:** sales-us@backbase.com

at once, and puts the main unit of work of the thread inside the call() method. The source for the QueryThreadPool class is in Listing 6.

To make it easier to switch between the two threading models, a simple interface was extracted from the original QueryThread implementation named IsAConcurrentQueryThreadRunner, mandating the following methods: getResultSet(), getSQLException(), and isAlive(). See IsAConcurrentQueryThreadRunner.java below.

```
package net.sourceforge.concurrentQuery.article.concurrent;

import java.sql.ResultSet;
import java.sql.SQLException;

public interface IsAConcurrentQueryThreadRunner {

    public ResultSet getResultSet();
    public SQLException getSQLException();
    public boolean isAlive();
}
```

This interface is used on the ConcurrentHashMap lists that hold the references to the running query threads. Now, it's possible to change a few references of the QueryThread to the QueryThreadPool and vice-versa to switch between the two threading models. Of course, a factory to create the threading model based on a properties file would be more efficient, but outside the immediate scope of our discussion. The entire source for the ConcurrentQueryThreadImpl class is in Listing 7.

A Second, More Robust Implementation

To demonstrate use further, I've put together a more elaborate implementation of this pattern that builds a large object from real database queries. This database has one table that lists cities with large populations, their districts (or states), and the countries in which they reside. For this example, I have built a single object that contains a list of countries that have more than 75 cities. The CountryList object contains a list of its districts, each district contains a list of its cities. All of this is in one big object. Once it's built, the results are printed. Below is Partial output from printing the CountryList object.

```
=== stuff deleted ===

 Country Code: USA
          District: Alabama
                  city name: Birmingham, population: 242820
                  city name: Huntsville, population: 158216
                  city name: Mobile, population: 198915
                  city name: Montgomery, population: 201568
          District: Alaska
                  city name: Anchorage, population: 260283

=== stuff deleted -==
```

Once built, this object contains 11 countries, 350 districts each associated with its country, and 2,233 cities each associated with its district. I've implemented the solution using a concurrent query pattern that uses a factory to create a concurrent query object with the desired threading model (normal threads, callable thread pool, or runnable thread pool). Then I created a factory broker singleton class that reads the threading model, JDBC settings. and the number of connections from a properties file and invokes the proper factory to create the concurrent query object. If I use one connection, thus simulating a serialized approach, it takes about 30 seconds on average to construct this object (this doesn't include the amount of time needed to print the results). If I use two connections concurrently, the process of constructing the object takes only about 7.7 seconds. Using three connections gets the time down to 5.2 seconds. Your mileage may vary and you will eventually hit a point of diminishing returns where adding more concurrent connections won't improve performance.

Consider the CountryList domain class in Listing 8 that accepts an argument for the number of cities, builds a list of countries that have more than that number of cities, and then constructs a list of the districts in each country.

Note that the processResultSet method is defined in the ResolvableFromConcurrentQuery interface. Also, the DistrictList class, which is instantiated by the CountryList object, is a domain object that participates in concurrent queries and will invoke a CityList object, yet another concurrent query domain object. And all of this happens using threaded queries and queued queries on lists to manage them. Notice too that in this implementation that I've chosen to have the domain objects explicitly call the resolve() method of the ConcurrentQuery object rather than build a notification into the interface as the previous implementation did with the isReaped() method. The resolve() method waits for all the running threads and queued queries to complete before continuing. The tradeoff is whether or not it's more feasible to have each getter in the domain object check to be sure it's reaped or whether it's better to have the domain objects explicitly wait to be resolved.

So, in general, a concurrent query implementation will likely have a mechanism to invoke a SQL query without waiting for the SQL results, and a way to ensure that an object is properly built before it's used – either by having the business logic explicitly wait for all results to finish after invoking some concurrent queries, or by having the domain object itself recognize that it hasn't processed its SQL results and requests to wait for those results.

When To Use Concurrent Queries

I wouldn't propose using a concurrent query pattern as a general rule for all database access because of resource constraints, but I believe there are many applications that could benefit from occasional use. This pattern fits most easily with POJOs that already build and execute and process results for their own SQL queries. The following are characteristics of applications that might benefit:

- Database and server resources are adequate and the database server isn't already under duress.
- Your application is already using JDBC queries.
- Your application controls when queries are run and when the results are processed (e.g., not using an external tool for building, managing, and running queries).

- You're not having issues with the number of connections available to the database server.

If so, then it might be feasible to implement this pattern. Remember, you can always configure the number of queries allowed to run concurrently to one, essentially running your application as a regular serialized JDBC query/result model, if resource constraints become an issue.

Conclusion

Such a simple pattern can be implemented in a few hours and the results might help a project over some bumpy performance issues. A few items worth noting that didn't seem to fit in anywhere else:

- Concurrent queries don't have to be implemented using threads. Since most database servers are multithreaded themselves they usually return control back to the client after a query has been parsed and submitted while the database server works on the query. If you hold the connection then you can check for the results later without having to use threads (e.g., set a timeout to zero and check for a result). Of course, the threading approach is pretty efficient and I personally like that model better. While it's entirely possible to use JDBC and hold the connection without immediately processing the result, the de facto standard for Java/JDBC development, up to this point, has been to submit queries and process results in one operation. But, when using a language or platform whose threading package isn't trustworthy then this pattern can be implemented without threads. In a previous project, I implemented a variation of this pattern using C and ODBC without threads.
- If you access a singleton concurrent query implementation from threaded clients then you might need to synchronize methods or blocks strategically in the concurrent query singleton.
- I've never implemented this pattern with objects that insert, update or delete data, but I suppose it could be done. I've never implemented this pattern to participate in a transaction, but that too should be possible.
- Besides building query-intensive objects faster, another potential use for this pattern could be in improving front-end user response time by pre-fetching data. For instance, suppose that after a user logs in to your application, his likely next choice would be to pull a list of active orders, view a list of products, or view their account settings. Concurrent queries could be used to build objects for all three potential choices immediately after the user logs in. By the time the user decides on which option to choose, the domain objects would be immediately available, or at least closer to being available than if the object started to be constructed after the user made a choice. Of course, an expiration date on the object would be in order in case the user takes 30 minutes to make a choice. Sure, you might end up building an object that you don't use, but I've had several instances where the perceived user response time was more valuable than the application resources. I don't like fast food restaurants that have my burger made before I actually order it, but I'm not as picky about my data.

For More Information

All of the sources found here, plus the source for the implementation of the list of countries example is available on sourceforge.net. Since concurrent query is more of a pattern than a packaged solution, the project on sourceforge.net is just a sample implementation intended for perusal. Sources are available for download from http://sourceforge.net/projects/concurrentquery.

The SleepyObject (ant target: run-example1) and ConcurrentSleepyObject (ant target: run-example2) are found in the *article* package and use a Postgres database. View the readme for instructions on creating the sleep function in Postgres. Other database servers might have a built-in function (e.g., waitfor in MS SQL) that could be substituted.

The country list example (ant target: run-ModelDriver) uses a MySQL database server. The DDL and data to create the city table is included and instructions for loading are also in the readme file.

Listing 1: SleepyObject.java

```
package net.sourceforge.concurrentQuery.article.serialized;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

import net.sourceforge.concurrentQuery.test.pool.JDCConnectionDriver;

public class SleepyObject {
    private String jdbcDriver = "org.postgresql.Driver";
    private String jdbcURL = "jdbc:postgresql://localhost:5432/
test?user=postgres";
    private String jdbcPoolURL = "jdbc:jdc:jdcpool";
    private String jdbcUser = "postgres";
    private String jdbcPasswd = "postgres";
    private int value = 0;

    public SleepyObject(int sleepSeconds) throws SQLException {
        try {
            Class.forName(jdbcDriver).newInstance();
        } catch (Exception e) {
            throw new SQLException(e.getMessage());
        }
        try {
            new JDCConnectionDriver(jdbcDriver, jdbcURL, jdbcUser,
                    jdbcPasswd);
        } catch (Exception e) {
            throw new SQLException(e.getMessage());
        }
        create(sleepSeconds);
    }
```

```
    private void create(int sleepSeconds) throws SQLException {
        Connection connection = DriverManager.getConnection(jdbcPoolURL,
                                                jdbcUser, jdbcPasswd);
        Statement statement = connection.createStatement();
        String sql = "select sleep(" + sleepSeconds + ")";
        System.out.println("query is: " + sql);
        if (statement.execute(sql)) {
            ResultSet resultSet = statement.getResultSet();
            if (resultSet.next()) {
                value = resultSet.getInt(1);
            }
            connection.close();
        }
    }

    public int getValue() {
        return value;
    }
}
```

Listing 2: ConcurrentQueryThreadImpl.java properties and initialization

```
public class ConcurrentQueryThreadImpl {
    private String jdbcDriver = "org.postgresql.Driver";
    private String jdbcURL = "jdbc:postgresql://localhost:5432/
test?user=postgres";
    private String jdbcPoolURL = "jdbc:jdc:jdcpool";
    private String jdbcUser = "postgres";
    private String jdbcPasswd = "postgres";

    private static ConcurrentQueryThreadImpl instance = null;
    private final int numberOfConcurrentQueries = 5;
    private static ConcurrentHashMap<CanResolveAConcurrentQuery,
      String> queuedQueries;
    private static ConcurrentHashMap<QueryThread,
                            CanResolveAConcurrentQuery> runningThreads;

    public static ConcurrentQueryThreadImpl getInstance() throws
SQLException {
        if (instance == null) {
            instance = new ConcurrentQueryThreadImpl();
            queuedQueries = new ConcurrentHashMap<CanResolveAConcurrent-
                    Query, String>();
            runningThreads = new ConcurrentHashMap<QueryThread,
                                    CanResolveAConcurrentQuery>();
        }
        return instance;
    }

    private ConcurrentQueryThreadImpl() throws SQLException {
        try {
                Class.forName(jdbcDriver).newInstance();
        } catch (Exception e) {
                throw new SQLException(e.getMessage());
        }
        try {
                new JDCConnectionDriver(jdbcDriver, jdbcURL, jdbcUser,
                    jdbcPasswd);
        } catch (Exception e) {
                throw new SQLException(e.getMessage());
        }
    }

== stuff deleted ==
```

Listing 3: ConcurrentSleepyObject.java

```
package net.sourceforge.concurrentQuery.article.concurrent;
import java.sql.ResultSet;
import java.sql.SQLException;

public class ConcurrentSleepyObject implements CanResolveAConcurrentQuery {

    private int value = 0;
    private boolean reaped = false;

    public ConcurrentSleepyObject(int seconds) throws SQLException {
        create(seconds);
    }

    private void create(int sleepSeconds) throws SQLException {
        ConcurrentQueryThreadImpl.getInstance().runQuery("select sleep("
                                            + sleepSeconds + ")", this);
    }

    public int getValue() throws SQLException {
        if (!reaped) {
            ConcurrentQueryThreadImpl.getInstance().waitForQueriesTo-
                    Complete();
        }
        return value;
    }

    // implemented method to process JDBC results
    public boolean processResultSet(ResultSet rs) throws SQLException {
        if (rs.next()) {
            value = rs.getInt(1);
            return true;
        } else {
            return false;
        }

    }

    public void setReaped(boolean isReaped) {
        this.reaped = isReaped;
    }


}
```

Listing 4: Code fragment of ConcurrentQueryImpl.java

```
public class ConcurrentQueryThreadImpl {
…
public void runQuery(String query, CanResolveAConcurrentQuery domainObject)
throws SQLException {
        Connection connection = DriverManager.getConnection(jdbcPoolURL);

        // reap any results from completed threads, if any
        reapCompletedThreads();

        // before we start a QueryThread for this query, let's submit any
        queries
        // that have already been queued
        while (!queuedQueries.isEmpty()
                && runningThreads.size() < numberOfConcurrentQueries) {
            CanResolveAConcurrentQuery queuedDomainObject
                  = (CanResolveAConcurrentQuery)queuedQueries.keySet().
                     toArray()[0];
            String queuedQuery = queuedQueries.get(queuedDomainObject);
            queuedQueries.remove(queuedDomainObject);
            runningThreads.put(new QueryThread(connection, queuedQuery),
                                queuedDomainObject);
        }
```

```
        // now, either start a thread for this query or add it to the
queued queries.
        if (runningThreads.size() < numberOfConcurrentQueries) {
            runningThreads.put(new QueryThread(connection, query), domain-
Object);;
        } else {
            queuedQueries.put(domainObject, query);
        }
    }

    public void waitForQueriesToComplete() throws SQLException {
        while (!queuedQueries.isEmpty() || !runningThreads.isEmpty()) {
            do {
                reapCompletedThreads();
            } while (!runningThreads.isEmpty());

            while (!queuedQueries.isEmpty()
                    && runningThreads.size() < numberOfConcurrentQueries)
{
                CanResolveAConcurrentQuery queuedDomainObject
                     = (CanResolveAConcurrentQuery)queuedQueries.key-
Set().toArray()[0];
                String queuedQuery = queuedQueries.
get(queuedDomainObject);
                queuedQueries.remove(queuedDomainObject);
                runningThreads.put(new QueryThread(
                                            DriverManager.getConnection(j
dbcPoolURL),
                                            queuedQuery),
                                    queuedDomainObject);
            }
        }
    }
…
```

Listing 5: Implementation of a QueryThread private class in ConcurrentQueryThreadImpl

```
public class ConcurrentQueryThreadImpl {

...

private class QueryThread extends Thread implements IsAConcurrentQueryThre
adRunner {

        private Connection connection;
        private String query;
        private ResultSet resultSet;
        private SQLException sqlException;

        private QueryThread() {}
        public QueryThread(Connection connection, String query) {
            System.out.println("(QueryThread) query is: " + query);
            this.connection = connection;
            this.query = query;
            start();
        }

        public void run() {
            Statement statement = null;
            try {
                statement = connection.createStatement();
               if (statement.execute(query)) {
                    resultSet = statement.getResultSet();
                    connection.close();
               }
            } catch (SQLException e) {
                sqlException = e;
            }
        }
        public SQLException getSQLException() {
            return sqlException;
        }
        public ResultSet getResultSet() {
            return resultSet;
        }

    };
...
```

Listing 6: QueryThreadPool private class

```
private static ExecutorService executor = Executors.newFixedThreadPool(numb
erOfConcurrentQueries);
    private class QueryThreadPool implements IsAConcurrentQueryThreadRunne
r, Callable {
        private Connection connection;
        private String query;
        private ResultSet resultSet;
        private SQLException sqlException;
        private Future<?> future;

        private QueryThreadPool() {}
        public QueryThreadPool(Connection connection, String query) {
                System.out.println("(QueryThreadPool) query is: " +
                     query);
                this.connection = connection;
                this.query = query;
            future = executor.submit(this);
        }
        public ResultSet call() throws SQLException {
            Statement statement = null;
            try {
                statement = connection.createStatement();
               if (statement.execute(query)) {
                    resultSet = statement.getResultSet();
                    connection.close();
               }
            } catch (SQLException e) {
                sqlException = e;
                throw new SQLException(e.getMessage());
            }
            return resultSet;
        }

        public SQLException getSQLException() {
            return sqlException;
        }

        public ResultSet getResultSet() {
            return resultSet;
        }

        public boolean isAlive() {
            return !future.isDone();
        }


    };
```

Listing 7: ConcurrentQueryThreadImpl.java

```
package net.sourceforge.concurrentQuery.article.concurrent;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
```

```
import java.sql.SQLException;
import java.sql.Statement;
import java.util.concurrent.Callable;
import java.util.concurrent.ConcurrentHashMap;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;

import net.sourceforge.concurrentQuery.test.pool.JDCConnectionDriver;

public class ConcurrentQueryThreadImpl {
    private String jdbcDriver = "org.postgresql.Driver";
    private String jdbcURL = "jdbc:postgresql://localhost:5432/
test?user=postgres";
    private String jdbcPoolURL = "jdbc:jdc:jdcpool";
    private String jdbcUser = "postgres";
    private String jdbcPasswd = "postgres";

    private static ConcurrentQueryThreadImpl instance = null;
    private static final int numberOfConcurrentQueries = 5;
    private static ConcurrentHashMap<CanResolveAConcurrentQuery, String>
          queuedQueries;
    private static ConcurrentHashMap<IsAConcurrentQueryThreadRunner,
                            CanResolveAConcurrentQuery> runningThreads;

    public static ConcurrentQueryThreadImpl getInstance() throws
SQLException {
        if (instance == null) {
            instance = new ConcurrentQueryThreadImpl();
            queuedQueries = new ConcurrentHashMap<CanResolveAConcurrent-
                    Query, String>();
            runningThreads = new ConcurrentHashMap<IsAConcurrentQuery-
                    ThreadRunner,
                                        CanResolveAConcurrentQuery>();
        }
        return instance;
    }

    private ConcurrentQueryThreadImpl() throws SQLException {
        try {
                Class.forName(jdbcDriver).newInstance();
        } catch (Exception e) {
                throw new SQLException(e.getMessage());
        }
        try {
                new JDCConnectionDriver(jdbcDriver, jdbcURL, jdbcUser,
                    jdbcPasswd);
        } catch (Exception e) {
                throw new SQLException(e.getMessage());
        }
    }

    public void runQuery(String query, CanResolveAConcurrentQuery domain
      Object)

 throws SQLException {
        Connection connection = DriverManager.getConnection(jdbcPoolURL);

        // reap any results from completed threads, if any
        reapCompletedThreads();

        // before we start a QueryThread for this query, let's submit any
                    queries
        // that have already been queued
        while (!queuedQueries.isEmpty()
                && runningThreads.size() < numberOfConcurrentQueries) {
            CanResolveAConcurrentQuery queuedDomainObject
                    = (CanResolveAConcurrentQuery) queuedQueries.keySet().
                                    toArray()[0];
            String queuedQuery = queuedQueries.get(queuedDomainObject);
            queuedQueries.remove(queuedDomainObject);
            runningThreads.put(new QueryThread(connection, queuedQuery),
                                    queuedDomainObject);
        }

        // now, either start a thread for this query or add it to the
        queued queries.
        if (runningThreads.size() < numberOfConcurrentQueries) {
            runningThreads.put(new QueryThread(connection, query), domain
                    Object);;
        } else {
            queuedQueries.put(domainObject, query);
        }
    }

    public void waitForQueriesToComplete() throws SQLException {
        while (!queuedQueries.isEmpty() || !runningThreads.isEmpty()) {
            do {
                reapCompletedThreads();
            } while (!runningThreads.isEmpty());

            while (!queuedQueries.isEmpty()
                    && runningThreads.size() < numberOfConcurrentQueries)
{
                CanResolveAConcurrentQuery queuedDomainObject
                    = (CanResolveAConcurrentQuery)queuedQueries.keySet().
                            toArray()[0];
                String queuedQuery = queuedQueries.
get(queuedDomainObject);
                queuedQueries.remove(queuedDomainObject);
                runningThreads.put(new QueryThread(
                                        DriverManager.getConnection(j
dbcPoolURL),

  queuedQuery),
                                    queuedDomainObject);
            }
        }
    }

    private void reapCompletedThreads() throws SQLException {

        for (IsAConcurrentQueryThreadRunner queryThread : runningThreads.
                    keySet()) {
            if (! queryThread.isAlive()) {
                CanResolveAConcurrentQuery domainObject
                    = runningThreads.get(queryThread);
                domainObject.setReaped(true);
                if (queryThread.getSQLException() != null) {
                    runningThreads.remove(queryThread);
                    throw new SQLException(queryThread.getSQLException().
                            getMessage());
                } else {
                    domainObject.processResultSet(queryThread.getResult
                            Set());
                    runningThreads.remove(queryThread);
                }
            }
        }
    }

    private class QueryThread extends Thread implements IsAConcurrentQuery
ThreadRunner {

        private Connection connection;
        private String query;
        private ResultSet resultSet;
        private SQLException sqlException;
```

Introducing JReport Live™

JReport® 8

Bring Your Reports to Life

JReport Live is the only solution to empower your Java application with operational business intelligence. Now your application can support both operational reporting and analytics – making users and developers happy.

Powerful Analytics Based on Dynamic Cube Technology

For the first time your users can enjoy ad hoc reporting right from the application, working with the most current operational data. Users can now slice-and-dice data, expand/collapse groups, pivot and drill up/down/across any report. The power of JReport Live is based on our Dynamic Cube Technology. JReport Dynamic Cubes are easy to define and are instantly created with no overhead. Information is always current and presented in the context of the application.

An Enterprise-class Foundation for Operational BI

JReport Live is built on our powerful operational reporting solution, delivering all of the benefits of scalability, performance and security of JReport 8. Plus, you can combine reports into report sets for better performance, easy scheduling and fast deployment. The pipeline mode allows users to start viewing reports before generating a full report. JReport 8 can integrate with any Java EE application and any security scheme. In addition, JReport runs reports on demand, on a schedule or by event triggers.

Build Complex Precise Reports Quickly and Easily

Only JReport 8 is not constrained by a rigid report layout, it lets you mix and match report components and control precisely how they are presented on the page. With multimedia objects, Web controls and Web forms to further enhance reports, your users will be happy to work with the highly functional, interactive JReport Live reports.

Download your version of JReport 8 with JReport Live or call 240-477-1000 today.

Bring your Reports to Life with JReport Live Server

For more information, visit www.jinfonet.com/live

JReport®
JINFONET SOFTWARE

```
    private QueryThread() {}
    public QueryThread(Connection connection, String query) {
        System.out.println("(QueryThread) query is: " + query);
        this.connection = connection;
        this.query = query;
        start();
    }

    public void run() {
        Statement statement = null;
        try {
            statement = connection.createStatement();
           if (statement.execute(query)) {
               resultSet = statement.getResultSet();
               connection.close();
           }
        } catch (SQLException e) {
            sqlException = e;
        }

    }
    public SQLException getSQLException() {
        return sqlException;
    }
    public ResultSet getResultSet() {
        return resultSet;
    }

};

private static ExecutorService executor
           = Executors.newFixedThreadPool(numberOfConcurrentQueries);

private class QueryThreadPool implements IsAConcurrentQueryThread-
  Runner, Callable {
    private Connection connection;
    private String query;
    private ResultSet resultSet;
    private SQLException sqlException;
    private Future<?> future;

    private QueryThreadPool() {}
    public QueryThreadPool(Connection connection, String query) {
            System.out.println("(QueryThreadPool) query is: " +
                query);
            this.connection = connection;
            this.query = query;
        future = executor.submit(this);
    }
    public ResultSet call() throws SQLException {
        Statement statement = null;
        try {
            statement = connection.createStatement();
           if (statement.execute(query)) {
               resultSet = statement.getResultSet();
               connection.close();
           }
        } catch (SQLException e) {
            sqlException = e;
            throw new SQLException(e.getMessage());
        }
        return resultSet;
    }

    public SQLException getSQLException() {
        return sqlException;
    }

    public ResultSet getResultSet() {
        return resultSet;
    }

    public boolean isAlive() {
        return !future.isDone();
    }


};
}
```

Listing 8: CountryList.java

```
package net.sourceforge.concurrentQuery.domain.model;

import net.sourceforge.concurrentQuery.domain.ResolvableFromConcurrentQuery;
import net.sourceforge.concurrentQuery.query.ConcurrentQueryFactoryBroker;
import net.sourceforge.concurrentQuery.query.ConcurrentQueryInterface;

import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Map;
import java.util.TreeMap;

public class CountryList  implements ResolvableFromConcurrentQuery {

    private ConcurrentQueryInterface cq = null;
    private Map<String, DistrictList> mapOfCountries = new TreeMap<String,
          DistrictList>();

    private CountryList(){}
    public CountryList(int numberOfCities) throws SQLException {

        this.cq = ConcurrentQueryFactoryBroker.getInstance().create-
                    ConcurrentQuery();

        cq.addQuery("select countrycode from city "
                    + "group by countrycode "
                    + "having count(*) > " + numberOfCities, this);
        cq.resolve();

        for (String country : mapOfCountries.keySet()) {
            mapOfCountries.get(country).fetchDistrictInfo(country);
        }

        cq.resolve();
    }

    public boolean processResultSet(ResultSet rs) throws SQLException {

        if (rs != null) {
            while (rs.next()) {
                mapOfCountries.put(rs.getString("countrycode"), new
                    DistrictList());

            }
        }

        return true;
    }

    public Map<String, DistrictList> getMapOfCountries() {
        return mapOfCountries;
    }
}
```

YOUR IDEA IS VISIONARY.
YOUR PLATFORM IS NOT.

MOBILITY. INTEROPERABILITY. INTELLIGENCE.

Meet the next-generation platform for distributed and edge computing– the only one that is completely platform independent.

VOYAGER Edge™

Your apps may be visionary, but they're dependent upon enterprise connectivity and distribution. When choosing a development platform, you need a solution that works everywhere- even where connectivity is limited. Voyager Edge™ is a proven, next-generation platform that uses intelligent mobile agent technology to solve mobility and performance issues in highly distributed environments. Faster than RMI, more reliable than client-server systems, Voyager agents can create ad-hoc networks that continue to work when they can't access the enterprise.

Voyager Edge gives architects maximum flexibility to freely develop dynamic, intelligent and decentralized applications in .NET and Java, on the devices and servers they need to target. These applications can run on centralized and edge devices, moving from one device to another while processing the same application. This unifying, next-generation platform can be used to gather, filter, analyze and distribute knowledge rapidly over today's increasingly heterogeneous wired and wireless networks.

Visit Recursion Software at www.recusionsw.com to make your vision a reality.

Download a free trial or intelligent mobile agent white paper at www.recursionsw.com.

Call 800.727.8674 to speak with an engineer.

RECURSION SOFTWARE, Inc.

The Development Power of Open Source AJAX Tooling

Bridging the gap between the development environment and the deployed browser environment

by Kevin Sawicki

Understanding the complexity of AJAX at the browser level is critical to refining and debugging rich AJAX applications that leverage Web technologies such as JavaScript, Cascading Style Sheets (CSS), and XMLHttpRequests. Adding a third-party AJAX runtime heightens the complexity and sufficient browser tooling becomes critical when attempting to build a rich Internet application around existing libraries. The Eclipse AJAX Toolkit Framework (ATF) provides both a multi-faceted set of browser tooling features as well as support for integrating and building on existing AJAX runtimes.

IBM has a long history of working with the open source community and supporting innovation and collaboration. The Eclipse ATF project is one of 150 that IBM is currently participating in and actively making technology contributions for various enhancements. IBM's goal around ATF is to accelerate the industry's acceptance of AJAX by offering it a complete open source AJAX IDE. The project including plug-ins to the Eclipse development platform is in ongoing development efforts to provide exemplary tools for creating and debugging AJAX applications.

How ATF Works

ATF provides views into the CSS, XMLHttpRequests, and the Document Object Models (DOM) of a Web application along with any error, warning, or information messages generated by the browser. These views are linked to the currently loaded document in the browser and responds to selections make of individual DOM nodes. Inspection is crucial for finding and locating problems in rich AJAX applications and doing this inspection in the same environment as development allows rapid application development and deployment. ATF provides a capability to select a DOM node and unravel it to expose any JavaScript functions referenced in the markup, the CSS rules and properties currently applied, and the HTML source. This allows quick discovery of the style sheets and JavaScript functions being used by third-party AJAX widgets, increasing their flexibility and maintainability. Sifting through directories and documentation to find out how widgets are defining their look-and-feel is no longer needed since simply selecting the widget in the embedded browser will reveal the file and line number of the CSS or JavaScript code used. Developers can now maintain API-level abstraction from the toolkits used during development but then have access via inspection into the components of the rich AJAX application at the browser level when debugging a deployed application.

The DOM Source view in ATF provides both an inspection tool and an editor for viewing and also making changes to a node's DOM source. ATF enables the browser to become a canvas that enables modifications to the DOM to be re-injected into the browser and any functional or visual changes will be realized. The source tooling around the embedded browser provides live rendering of changes made to any node in the DOM document. This feature has interesting and empowering applications such as the ability to inject link and script nodes into the page's DOM and then make use of the loaded scripts when editing other nodes on the page such as controls or user interface elements. This feature lets AJAX be enabled on any Web site that can be loaded via a URL and allow mashups of locally created and deployed scripts with remote pages. This editing capability provides developers with a sandbox

> "The Eclipse ATF project is one of 150 that IBM is currently participating in to accelerate the industry's acceptance of AJAX by giving it a complete open source AJAX IDE"

Kevin Sawicki is an acting strategy engineer with IBM in the Emerging Internet Technology Group in Austin, Texas. His responsibilities include the strategic development and growth of AJAX technology to be contributed to the open source community.

ksawicki@us.ibm.com

> The DOM Source view in ATF provides both an inspection tool and an editor for viewing and also making changes to a node's DOM source"

for testing and exploring runtime toolkits as well as changing interactions dynamically within the browser. The DOM source tooling for the browser lets developers go so far as to develop rich AJAX applications completely inside the browser with no local project files on the system. The gap between written and rendered DHTML code is bridged as a side-by-side of runtime-specific source and browser source can be viewed and edited. AJAX application issues can now be resolved inside the browser, which shortens development iterations with changes only propagated back to application source files when the resulting application functionality is correct.

The use of Cascading Style Sheets is an integral part of rich Internet application development to achieve a consistent visual look-and-feel. Achieving the "desktop" application feel in a rich AJAX application means making use of CSS properties to create partial transparency when dragging an item into an online shopping cart. Style sheet management and refinement becomes tough for developers when mixing developed styles with existing styles in AJAX runtime toolkits. ATF provides a CSS view that's driven by selecting any DOM element and viewing all CSS rules currently applied to that element as well as any cascading that's occurring. This view allows any defined property for a rule to be changed and for properties to be added and deleted with any modifications affecting all nodes on the page using those style rules. The CSS tooling provides a feature to select a CSS rule and visually highlight all elements in the browser that currently have that property rule applied. This feature lets developers see what elements will be affected by changes to that property and exposes the differentiation of elements caused by cascading styles. The process of managing and refining style sheets becomes streamlined as developers can deploy applications and then tool the CSS live in the browser page until it's correct and then revert the changes made back to workspace project files using the diff computing of the CSS view that tracks all the CSS changes made since the page was loaded.

AJAX derives its power by using XMLHttpRequest (XHR) as a transport for asynchronous communication to other sites or Web servers. ATF provides a monitor of all XMLHttpRequests that occur for a page once it's loaded in the embedded browser. Developers can get the turnaround time for the request as well as the contents of the request and response and the target URL. The XHR monitor rounds out the necessary inspection tools for components used in rich AJAX applications.

Developers Can See the Benefits

Developers familiar with the Java debugging support inside Eclipse will see the same features in ATF for JavaScript debugging such as breakpoints, expression evaluation, and variable inspection. ATF brings the Eclipse Java debug experience to JavaScript-based applications with the same debugging behavior and functionality. ATF provides a debugging perspective that shows all the scripts currently loaded for a page launched in the embedded Mozilla browser. The script view can be used to inspect loaded scripts by opening them in an editor with support for breakpoint creation. Developers can easily discover and explore scripts used in the page by setting the necessary breakpoints and then interacting with the page to see what actions are handled by what scripts.

The debugger is unique in the sense in that its functionality is supported for both locally created and deployed applications and remote applications launched via URL. A Web application can be fully debugged with no local workspace files needed and JavaScript files can be opened remotely with support for adding remote breakpoints. Now testing a site simply requires ATF and a URL and from that breakpoints can be set for files, errors, exceptions, or on launch and the entire user interaction with an AJAX application can be stepped through. This lets developers explore runtime toolkits from the top down with a JavaScript stack frame that allows the correct level of inspection and evaluation to verify the correct use of widgets and AJAX support.

The complexity created by utilizing various Web technologies for building rich Internet applications requires tooling not just at the source level but at the browser level as well. The browser becomes a crucial development tool since inspection and tooling allow introspection and modification to take place for any page navigated to. This bi-directionality gives Web 2.0 developers and testers the necessary information and tooling needed to understand and resolve browser interactions. The AJAX Toolkit Framework seeks to bridge the gap between development environment and the deployed browser environment. ATF provides the necessary tooling around a Mozilla browser inside Eclipse and reveals, via inspection, all the necessary elements of the rich AJAX application for troubleshooting and debugging. ATF brings the Eclipse development experience to developers looking for a complete and sufficient tool for developing and debugging rich Internet applications. ATF is an open source project under the Eclipse Web Tools Platform. More information can be found at http://www.eclipse.org/atf.

instantiations CASE STUDY

BEA Uses RCP Developer™ to Improve Quality and to Save Time & Money

BEA relies heavily on the WindowTester™ component of the RCP Developer™ software from Instantiations to automate testing of GUI elements

"In terms of our cost of investment in the WindowTester tool, we believe the cost of the licensing is greatly worth the enhancement and efficiency gained so far."
—Bill Roth, VP of BEA Workshop Unit

This Case Study is available online at **www.instantiations.com/rcpdeveloper/resources/casestudy-bea.pdf**

BEA Systems and Instantiations have had a unique relationship while working in the Eclipse ecosystem. BEA is a strategic member of the Eclipse Foundation, has a representative on the Eclipse Board, and is an active participant in the Eclipse Web Tools project. Instantiations is a long time Eclipse member company, has a representative on the Eclipse Foundation Board, and has had a number of committers on Eclipse projects. The work delivered by Instantiations in the Eclipse Pollinate project provided early proof-of-concept technology for BEA's Eclipse-based tooling.

BEA Workshop makes developing Java applications easier by allowing Eclipse developers to quickly create, debug and test SOA components, Web Services, Web applications, BEA WebLogic Portal applications, and enable Service-Oriented Architecture (SOA) solutions. BEA Workshop is based on the Eclipse Open Source Integrated Development Environment (IDE). Eclipse includes a Rich Client Platform (RCP) layer that makes it easy to develop applications with extensive GUI capabilities.

BEA also utilizes other key Eclipse-based tools to develop and test their own applications. For example, BEA replies heavily on the WindowTester component of the RCP Developer software from Instantiations to automate the testing of GUI elements. RCP Developer is a software development product that accelerates the creation of Eclipse RCP applications by providing tools for constructing and testing graphical user interfaces, composing Help documentation and packaging rich client applications for deployment. We will explore how BEA is using the WindowTester component of RCP Developer in the remainder of this case study.

Development Challenge

Three years ago, Workshop's automated IDE-based testing was from a home-grown testing infrastructure. BEA is shifting its product release schedule so releases will be delivered in 1/6th the time previously required. According to Steve Tocco, BEA Workshop Director of Quality Assurance, "Our internal IDE testing had inadequate code coverage numbers with 1/20th the number of tests our current automated test suite contains. We determined that the amount of intermittent failures, the inadequate automated coverage, as well as the cost to implement tests was prohibitive in getting products to market quickly." The BEA Workshop group needed a better solution for automated testing of their application GUIs. They chose Instantiations RCP Developer and its innovative WindowTester functionality to meet this need.

Solution: Using WindowTester for Automated GUI Testing

Testing History

In addition to the in-house testing infrastructure used by BEA Workshop group, they previously used and evaluated other commercial off-the-shelf products. While some of the tools provided a quick-click and record of Swing tests, they would not address the needs raised as Workshop moved to the Eclipse infrastructure in 2004. They researched options and decided to initially create their own test suite using the open source Abbot SWT GUI testing framework. For the product release that shipped in July 06, they used the Abbot structure for testing.

Components of RCP Developer™ 2.0

- **SWT Designer™** automatically generates Java code from a powerful and intuitive visual designer
- **WindowTester™** records events and automatically generates GUI tests based on the JUnit standard
- **Help Composer™** streamlines the creation of documentation that is fully compatible with the Eclipse Help system
- **RCP Packager™** quickly automates deployment by generating a flexible SWT-based Installer

However, the team found that creating and running tests still did not meet their needs. The group decided that developing and maintaining their own test harness was not their core competency and they did not want to devote extensive resources to creating a test infrastructure. Tocco states, "To meet a rapid release schedule, the test suite must be optimized as we can't afford false negatives or instabilities drawing precious resources away from our deliverables. Even more, tests need to be authored rapidly as we try to find optimizations in our schedules while not compromising quality for our customers. We weren't getting that in any form before. Our team felt we weren't agile enough to meet the new release timelines—we simply don't have time now to rely on manual tests alone."

Moving to WindowTester for Automated GUI Testing

The BEA Workshop team began looking at tools that could automate the testing of GUI elements of their Eclipse-based applications. They chose RCP Developer from Instantiations and are converting their test suite to use its WindowTester component. They selected a commercial product even though there are open source alternatives available because of the tools' ease of use as well as Instantiations' support, responsiveness, and professional documentation. According to Bill Roth, VP of BEA Workshop Unit, "In terms of our cost of investment in the WindowTester tool, we believe the cost of the licensing is greatly worth the enhancement and efficiency gained so far."

How Testing Has Changed after Implementing WindowTester

When evaluating RCP Developer and WindowTester, the BEA Workshop for WebLogic team found that it took about two months to achieve a 40% line coverage in the product from a code coverage perspective. According to Tocco, "Since we started using WindowTester, tests that took two to three weeks to write previously can now be done in two to three days. It is much faster to get a test developed and running. In addition, we have a higher stability rate than before with a extraordinary pass rates in our automated regression tests. This is fantastic. It lets us spend our energy building the product, not chasing test issues." The BEA Workshop for WebLogic team also found that WindowTester can handle a heavy test load. The team currently runs over 300 tests. Some of the tests are quite broad, with multi-step lengthy testing scenarios. This is a 20 times increase over what was in the release two years ago for IDE automation.

Future: Full Migration to WindowTester

The BEA Workshop for WebLogic team plans a complete migration away from the old test harness for the IDE in six months. The team intends to use the WindowTester component of RCP Developer as the sole tool for automated IDE testing tool in Eclipse by the end of 2006.

Summary

Moving to Instantiations RCP Developer and its WindowTester functionality has allowed the BEA Workshop for WebLogic group to drastically cut the time it takes to generate new GUI tests. The move to WindowTester has saved the group both time and money allowing them to focus on developing their product rather than creating and maintaining a test infrastructure.

"Since we started using WindowTester, tests that took 2–3 weeks to write previously can now be done in 2–3 days."
—Steve Tocco, BEA Workshop Director of Quality Assurance

About BEA — *www.bea.com*

BEA Systems is a company founded in 1995 that specializes in enterprise infrastructure software, and has 77 offices in 37 countries. BEA Systems, Inc. is a world leader in enterprise infrastructure software, delivering powerful standards-based platforms for building enterprise applications and managing Service-Oriented Architectures even in heterogeneous IT environments. Customers depend on BEA Tuxedo®, WebLogic®, and AquaLogic™ product lines to reduce IT complexity, leverage existing resources, and speed the delivery of new services. BEA also provides support for Blended strategies that combine Open Source and commercial software to best suit the needs of business and IT. With over 15,000 customers including the majority of the Fortune Global 500, BEA provides the technology, solutions and services to help companies achieve a state of Business LiquidITy™ where enterprise assets are freed up to deliver maximum business value.

About Instantiations — *www.instantiations.com*

Instantiations, Inc. provides leading-edge software products, services and technologies for Eclipse, Java and Smalltalk. Instantiations offers professional development environments and software products that integrate seamlessly with the latest development platforms. Instantiations is a member of the Eclipse Foundation and offers a line of products for Eclipse, Rational Application Developer, IBM WebSphere Studio and MyEclipse. Based in Portland, Ore., Instantiations was founded in 1997 by a team of internationally recognized pioneers in the field of component software technology.

About Eclipse — *www.eclipse.org*

Eclipse is an open source community whose projects are focused on providing an extensible development platform and application frameworks for building software. Eclipse is an open source community whose projects are focused on providing a vendor-neutral open development platform and application frameworks for building software. **The Eclipse Foundation** is a not-for-profit corporation formed to advance the creation, evolution, promotion, and support of the Eclipse Platform and to cultivate both an open source community and an ecosystem of complementary products, capabilities, and services.

What Is SDO?

Part One: The value of many of the facets of SDO

by Kelvin Goodson
and Geoffrey Winn

Service Data Objects (SDOs) simplify and unify Service Oriented Architecture (SOA) data access and code.

SDO complements the strength that SCA (Service Component Architecture) offers for simplifying development of SOA-based solutions. SCA handles the composition of service networks and SDO focuses on simplifying data handling. These technologies are getting significant support in the industry. The development of the SDO and SCA specifications is in the hands of the Open Service Oriented Architecture collaboration (http://www.osoa.org) and open source implementations of these specifications are being developed in the Apache Tuscany incubator project (http://incubator.apache.org/tuscany).

In this two-part article we use a scenario to demonstrate the value of many of the facets of SDO.

Some History

The first SDO specification was published in November 2004 as a result of collaborative between IBM and BEA. The Eclipse Foundation developed an open source implementation of this SDO 1 specification. SDO primarily addressed the lack of general applicability of the existing technologies such as JAXB and JDO. Around that time Microsoft entered this space with ADO.NET, offering a slightly different technical perspective. The SDO 2.0.1 specification appeared late in 2005 and is continuing to evolve, with wider industry involvement; at the time of writing revision 2.1 is imminent and revision 3.0 is in the pipeline.

The Advantages of SDO

SDO provides flexible data structures that allow data to be organized as graphs of objects (called data objects) that are composed of properties. Properties can be single or many valued and can have other data objects as their values. A data object can maintain a change summary of the alterations made to it, providing efficient communication of changes and a convenient way to update an original data source. SDO naturally permits disconnected data access patterns with an optimistic concurrency control model.

SDO offers a convenient way to work with XML documents. SDO implementations provide helpers to populate a data graph from both XML documents and relational databases and to read SDO metadata from an XML Schema Definition (XSD). Data objects can be serialized to XML and the metadata can be serialized to an XSD file (see Figure 1).

Data Objects can be introspected using the SDO metadata API to get information about types, relationships, and constraints.

SDO delivers unified and consistent access to data from heterogeneous sources. This provides both a simple programming model for the application programmer and lets tools and frameworks work consistently across those heterogeneous data sources.

SDO offers a single model for data across the enterprise.

The diagram below shows a WebUI client accessing data from a variety of sources, mediated by SDO. Web applications typically operate in a semi-connected fashion and rely on optimistic-concurrency. SDO is well suited to this environment, where data can be manipulated remotely and then a summary of the changes can be delivered back to the data sources (see Figure 2).

The following sections will introduce SDO in more detail.

A Scenario

This example is based on an imaginary project inspired by some real-world scenarios. A hypothetical group of universities, hospitals, and companies have embarked on a long-term collaboration to study some family of diseases that has both a genetic and environmental component. They will need to exchange the medical histories of the people they're treating and studying, and also exchange the medical histories of relatives. The data will likely come from disparate sources; basic patient data will probably be in a relational database; data from medical investigations conducted as part of this research project will be in XML documents; other medical data may come from less well known formats or custom sources. The *amount* of data about any given person will vary greatly. A long-

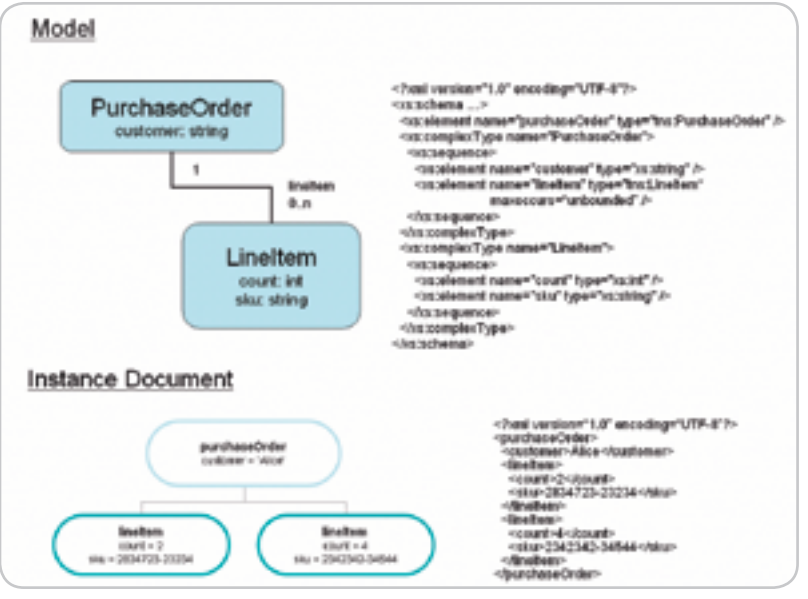


Figure 1

Kelvin Goodson is based at IBM Hursley in the UK as part of the Open Source SOA team. He is a committer to the Apache Tuscany incubator project, and works primarily on development of the Tuscany Java implementation of SDO. He gained a Ph.D. in image analysis and artificial intelligence in 1988, and has previously worked in the areas of medical imaging, weather forecasting and messaging middleware.

kelvin_goodson@uk.ibm.com

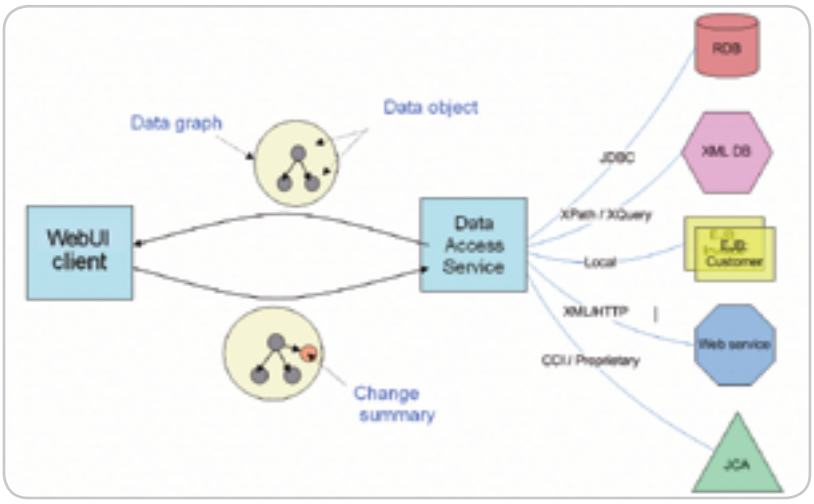


Figure 2

standing patient may come with an extensive medical history. A relative might have little beyond name and relationship. This data has to be assembled into a coherent manageable whole, and SDO is an attractive option for representing a complicated mix of data about each person and potentially maintain a graph of such entities. For this example, we can't even assert that the graph is a (family) tree because with adoption, re-marriage, fertility treatment, and so on, one person's associations with others can be quite intricate.

The various institutions involved may not want to give unrestricted access to their data sources, although they've agreed to supply pieces of it as needed. A hospital may be willing to provide the medical data associated with one patient as part of an investigation, but they won't permit open access to their entire patient record database. Similarly, a company will want to limit access to commercially sensitive material. SDO provides a convenient way for the owner of the data to deliver to outsiders a subset of that data of their own choosing.

We'll now show some of the key values of SDO through this scenario.

To illustrate where an SDO feature helps, consider a scenario where a hospital refers a patient to a university for further investigation. Relevant data will have to flow from the hospital to the university, and it may well come from a variety of different sources. Assume that name, age, records of visits, and so forth comes from an SQL database, while specific medical data (the results of tests) are in XML documents. Using standard SDO features it's straightforward for the hospital to combine these various sources into a data object and send that, letting users of the data access it via SDO's unified API.

The university does whatever it does with the patient, and then updates the SDO and sends it back. The change history in the SDO lets the hospital apply the updates to its various data repositories without the university ever needing to know the detail of those repositories.

It's unlikely that these updates will clash with other updates made independently by the hospital, but if they do, the use of an SDO change summary ensures that this is detected and sorted out (probably manually in this case). The software component responsible for moving data between the data source (for example, a relational database in this case) and SDO is called a Data Access Service (DAS). A DAS can typically also handle conflicting updates.[1]

Using SDO as the data exchange format makes the system tolerant of the considerable variation to be expected in such a loosely coupled system. It's inevitable that, sooner or later, the versions of the applications that are sending and receiving data will get out-of-step. In fact, this may be usual. However, the fact that an SDO can arrive with its own metadata means that an older application can always retrieve what it wants from a newer (and presumably richer) input SDO – ignoring anything that it doesn't recognise. In the reverse case, a newer application can similarly recognise that the information it has received is from an older version and compensate accordingly.

In the previous scenarios, we've concentrated mostly on XML and SQL data sources. Now, let's suppose that in one hospital the results from the biochemistry lab are delivered in HL7 message format. This message format is widely used in the healthcare industry but is virtually unknown outside it, and so there's no off-the-shelf way to read such messages into an SDO. At this point there are several choices. We could use some broker-style product to reformat the HL7 into XML and then read it into an SDO or we could pay someone to write a new DAS that would populate an SDO directly from HL7. Since our collaborators are using an open source implementation of SDO, however, they opt to write their own DAS and donate it to the Apache Software Foundation's Tuscany project.

Other approaches exist to linking these various organisations, such as putting some software intermediary in the middle and using that to convert the data as needed. To do so though requires knowledge of all the possible input and output formats and how to convert between them. In such a loose collaboration there simply is no such central authority.

We now turn our attention to presenting the details of SDO using some code fragments.

Creating Types

In SDO data objects have a type so the first step in presenting our example is to construct the types we're going to use. We have several choices here. The first choice we face is whether to generate static Java classes that represent the types or whether to build them dynamically.

In a situation where the type system is stable and well understood then generating static types leads to simpler, more natural coding. For example, with generated types we'd be able to code something like…

```
person.getPatientName()
```

as opposed to

```
person.getString("patientName")
```

Statically generated types also offer the possibility for the programmer to code to the generated interface without knowing the SDO API. The corollary of this is that when the type system isn't well known or might change then the dynamic SDO API may be more suitable. Choosing to use statically generated types has the advantage that the whole

Geoffrey Winn is based at IBM Hursley in the UK, as part of the Open Source SOA team. He is a member of the SDO specification group and currently works on development of the Apache Tuscany C++ implementation of SDO. He has degrees in Mathematics and Computation, and has previously worked in the areas messaging and brokering middleware.

gwinn@uk.ibm.com

SDO dynamic API is still available to the programmer for handling less common operations.

We're going to focus on dynamically building types, since it naturally leads to exploring more of the SDO API, but bear in mind that if we were to use the option of generating static classes we have the power of SDO operating behind the scenes while using Java method calls that are no more than JavaBean getters and setters.

An option when defining types dynamically might be to use the facilities of an existing DAS, which, for example, could convert from a database schema; we could also use SDO's XSDHelper to read an XSD and build SDO types from it. The SDO specification provides a way to create types dynamically, however, it depends on knowing the SDO API, which we haven't seen yet! To simplify this example, we'll use an extension from Apache Tuscany, which lets type definitions be built without knowing the SDO API.

```
Type personType = SDOUtil.createType(
      TypeHelper.INSTANCE, "www.example.
         org", "Person", false);
```

The net result of this line of code is the creation of an empty SDO type called "Person" scoped by the URI "www.example.org" and when completed, this type can be used to instantiate data objects. We can now add properties to the type and set its characteristics. Every property has a type, and we can make use of SDO's built-in types (in this case a string) to build our model.

```
Type stringType = TypeHelper.INSTANCE.
   getType("commonj.sdo", "string");
```

We use this type to add a "name" property to our "person" type.

```
SDOUtil.createProperty(personType,
   "name", stringType);
```

In our example scenario we can't know in advance all the information we might want to associate with a person. By making the type open we permit data objects having this type to carry additional properties that aren't defined as part of this type.

```
SDOUtil.setOpen(personType, true);
```

We could continue building this type in this way, however, it would rapidly become tedious. An alternative approach is to use SDO's XSDHelper to build a type system by reading XML schema definitions (see Figure 1).

The XSD for our type definitions is shown in Listing 1. Just as in our previous code example, we've made the Person type in the XSD open by using the "xsd:any." There are other interesting aspects of this schema that we'll develop as the example unfolds.

We can read this schema using SDO's XSDHelper and then we have three new types, Person, Relative, and PersonSet, available to us that can be used to create data objects.

```
File inputFile =
   new File("Person.xsd").getAbsoluteFile();
InputStream inputStream =
   new FileInputStream(inputFile);
List schemaTypes = xsdHelper.define(
   inputStream,
   inputFile.toURI().toString());
```

Building a Graph

These new types are all scoped within the URI www.example.org/people. Now that we have some types defined we're in good shape to create some data objects, so the first thing we do is to use the SDO DataFactory and our Person type to construct an SDO DataObject representing an instance of Person.

```
DataObject person1 = DataFactory.
   INSTANCE.create("www.example.org/
   people", "Person");
```

We know from the XSD that the Person type has "id," "name," and "gender" properties so we can set values for these properties as follows.

```
person1.setString("id", "1");
person1.setString("name", "Joe Johnson
      Snr.");
person1.setString("gender", "male");
```

We can begin building a graph by adding this person to a set of "referrals" in a test, i.e., the set of people who have been referred by a medical practitioner because they've exhibited symptoms or are related to an existing patient. We do this by creating a new DataObject of type PersonSet.

```
DataObject referrals = DataFactory.
   INSTANCE.create("www.example.org/
   people", "PersonSet");
```

The "people" property of the referrals DataObject is defined in the XSD as many valued and is therefore accessed via the getList method.

```
referrals.getList("people").
   add(person1);
```

Containment

Something to note here is that we have created a containment relationship between the person1 DataObject and the referrals DataObject. SDO treats containment relationships in a special way. In every data graph each data object apart from the root has a link to exactly one parent data object that is its container. The corresponding property from the parent to the child is marked as a containment property. So the containment relationships form a strict hierarchical structure whereby every data object in a graph is reachable by traversing only the containment links. To enable SDO to describe richer, more diverse graph structures, non-containment links can also be included into the graph. These non-containment links permit arbitrary references across the containment hierarchy.

SDO has behavior built in to preserve the containment hierarchy constraint; so, for example, if you assign a data object into a graph as the value of a containment property then any existing value at that location in the graph will be automatically displaced (the value of its parent property becomes null and the orphaned data object, along with any children it might have, becomes a separate data graph). In addition, if the data object that you're assigning to the graph is already contained in a data graph then it will be automatically removed from that containment association before being assigned to its new location.

Let's take a look at an example. Suppose that the institutions running the tests have learned from the mistakes of previous such projects in which patients already diagnosed with a condition have inadvertantly received letters intended for new referrals and this caused unnecessary distress. This was seen as sufficiently important for them to engineer their data models to ensure that referrals and

patients were always disjoint sets. Listing 2 shows the XML schema used to model the test. It makes use of the schema shown earlier to model the people involved in the test.

This schema has a global element named "test," which is of type "Test," which contains three sets of people: referrals, patients, and relatives. We saw from the previous schema that a PersonSet contains a list of people so this arrangement of containment relationships ensures that if we assign a person who already exists in the set of referrals to be in the set of patients then by virtue of the preservation of containments constraint, that person is automatically removed from the referrals set by the SDO infrastructure. Let's see that in action.

```
DataObject test = DataFactory.INSTANCE.
create("www.example.org/MedicalTest", "Test");
    test.set("referrals", referrals);
```

At this point our entire test consists of a set of referrals with one member (we haven't yet initialized the set of patients or relatives so they take the default null value). Here's how it would serialize to XML.

```
<Test:test xmlns:Test="www.example.org/
MedicalTest">
  <referrals>
    <person gender="male" id="1" name=
      "Joe Johnson Snr">
      <dob>1 January 1950</dob>
    </person>
  </referrals>
</Test:test>
```

Having visited the hospital, Joe Johnson Sr. is found to have symptoms indicating one of the conditions that the test is investigating, so he's added to the set of patients.

```
DataObject patients = test.createData-
Object("patients");
patients.getList("person").add(person1);
```

As an aside, note how we created the patients DataObject in a more concise way this time. The "test" DataObject knows the type of its "patients" property and so we don't need to use DataFactory to look up the type and create an instance.

Having made this assignment of person1 to the set of patients, this is how the whole test serializes.

```
<Test:Test xmlns:Test="www.example.org/
MedicalTest">
  <referrals/>
  <patients>
    <person gender="male" id="1" name=
      "Joe Johnson Snr."/>
  </patients>
</Test:Test>
```

So as a result of this one assignment, the patient's set has one member and the referrals set has been reduced to zero.

In part two of this article, we'll examine more SDO features, including the use of open content, change summaries, and references between objects that express more than containment.

References

[1] Williams, K., Daniel, B. "SOA Web Services - Data Access Service", *Java Developer's Journal*, August 2006, http://java.sys-con.com/read/260053.html

Listing 1

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
   targetNamespace="www.example.org/people"
   xmlns:sdo="commonj.sdo"
       xmlns:sdoxml="commonj.sdo/xml"
   xmlns:tns="www.example.org/people">

   <import namespace="commonj.sdo/xml" schemaLocation="sdoXML.xsd" />

   <complexType name="Person">
           <sequence>
                     <element name="dob" type="date"/>
                     <element name="relative" maxOccurs="unbounded"
type="tns:Relative"/>
                     <any namespace="##other" processContents="lax"
maxOccurs="unbounded"/>
           </sequence>
           <attribute name="id" type="ID"/>
           <attribute name="name" type="string"/>
       <attribute name="gender" type = "tns:Gender"/>
   </complexType>

   <complexType name="Relative">
       <attribute name="target" type="IDREF" sdoxml:propertyType="tns:
Person" use="required"/>
       <attribute name="relationship" type="string" />
       <attribute name="genetic" use="optional" type="boolean"/>
   </complexType>

   <complexType name="PersonSet">
           <sequence>
                     <element name="people" type="tns:Person"
maxOccurs="unbounded"/>
           </sequence>
   </complexType>

   <simpleType name="Gender">
       <restriction base="string">
             <enumeration value="male" />
             <enumeration value="female" />
      </restriction>
   </simpleType>

</schema>
```

Listing 2

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
   xmlns:people="www.example.org/people" xmlns:sdo="commonj.sdo"
   xmlns:sdoxml="commonj.sdo/xml"
   xmlns:tns="www.example.org/MedicalTest"
   targetNamespace="www.example.org/MedicalTest">

   <import namespace=»www.example.org/people»
           schemaLocation="People.xsd" />

   <element name="test" type="tns:Test" />
   <element name="condition" type="tns:Condition" />

   <complexType name="Test">
           <sequence>
                     <element name="referrals" type="people:PersonSet" />
                     <element name="patients" type="people:PersonSet" />
                     <element name="relatives" type="people:PersonSet" />
           </sequence>
   </complexType>

   <complexType name="Condition">
           <sequence>
                     <element name="diagnosed" type="date" />
           </sequence>
           <attribute name="name" type="tns:ConditionName" />
   </complexType>

   <simpleType name="ConditionName">
           <restriction base="string">
                     <enumeration value="Rigellian fever" />
                     <enumeration value="Vegan choriomeningitis" />
                     <enumeration value="Scrofungulus" />
                     <enumeration value="Panarr Syndrome" />
           </restriction>
   </simpleType>

</schema>
```

Reducing Maintenance Costs Through Systems Decoupling

by Konstantin Plotnikov and Vladimir Shraibman

Technological and functional decoupling

There are a number of reasons why we continuously change our enterprise applications, but sometimes it's hard to explain to the requester why another change would be so costly and time-consuming to implement. IT leaders who want to be fully prepared for inevitable changes in the corporate software infrastructure should do their best to avoid system coupling on both business and technological levels.

A modern enterprise information system (EIS) consists of hordes of applications bound together to support business activities in the enterprise. The degree of application coupling identifies the difficulty of making changes to them. Although coupling has been always used to describe the connections between application's components, it can be also used to describe relations between applications within the entire EIS.

Applications shouldn't be thought of as a system created forever. The technology is always improved, new data is collected, organizations are restructured, merges occur, and new laws and regulations are passed, all of which affect the EIS. High coupling is a trait that complicates systems maintenance and makes it more expensive to manage. Highly coupled solutions are more difficult to modify since changes made in one place cause changes to be made somewhere else. Although coupling is essential to the connected applications

that should have common assumptions about their interaction processes, we should typically work towards decreasing these levels of coupling.

The last technological response to the coupling and integration problems, is Service-Oriented Architecture (SOA).

There are two major aspects of coupling – technological and functional. They are addressed in different ways by different people.

- Technological coupling is the result of assumptions about transport protocols, data encoding schemes, authentication mechanisms, and so on. This aspect is the responsibility of system architects.
- Functional coupling comes from assumptions about the process of interaction and exchanged data. This aspect should be addressed by analysts and domain experts.

Technological Coupling

Technological coupling is caused by assumptions about technical aspects. The more the application knows about how interaction process is organized, the higher coupling is.

To become integrated, applications should speak the same language. This can be achieved by writing applications to use a single communication technology or by adapting them to each other using bridges, proxies, and other similar approaches. Within these methods, individual technological assumptions of particular solutions are replaced with a single standard set of assumptions. These assumptions are usually more high-level. For example, instead of specifying that input files should start with left brocket characters, we specify that it is XML text that conforms to an XML schema. These assumptions are usually supported by third-party libraries and tools, so we do not have to implement them explicitly in our applications. Thus Web service interaction assumptions are implemented by a respective runtime (in Microsoft .NET and Java EE, such a runtime is a part of corresponding platforms).

Today, the most popular solutions are based on Service-Oriented Architecture (SOA) and Enterprise Service Bus (ESB). There are many SOA/ESB products available on the market that helps integrate applications with each other with minimal changes and replaces development efforts with configuring efforts. Often the integration of two particular applications could not be foreseen at the time of

Konstantin Plotnikov is a senior architect at Axmor Software, Russian software development company. Constantine has been working with distributed applications since 1994, and has more than 14 years experience in the IT industry. During this tenure, he's been a programmer and software architect. He participated in JDO and JMI JCP expert groups.

cap@isg.axmor.com

> "A modern enterprise information system (EIS) consists of hordes of applications bound together to support business activities in the enterprise"

Enterprise AJAX is ICEfaces

- Create secure Rich Internet Applications (RIA)
- Develop in Java, not JavaScript
- Transform the User Experience

Visit www.ICEfaces.org to try it out!

RIA Solution for SOA

Rich User Experience

Create a new class of collaborative and dynamic enterprise applications. Unleash the unique power of Ajax Push Technology to deliver server-initiated, instantaneous presentation updates. Easily migrate existing JSP-based and traditional client-server applications to RIAs.

Standards-Based

Develop and deploy scalable RIAs in pure Java using an integrated Ajax framework complete with rich JSF-based components. Harness the power of Ajax and leverage the entire standards-based Java EE ecosystem of familiar tools and runtime environments.

Performance, Scale and Security

Deploy rich enterprise applications with advanced Ajax connection management for maximum application reliability. Seamlessly scale applications across clustered Java EE servers. Extend the existing web security model and avoid transferring sensitive business logic and data to the client browser.

ICESOFT
THE RICH WEB COMPANY

ICESOFT TECHNOLOGIES
Toll Free 1.877.263.3822
www.icesoft.com

> When problems of technical coupling are solved, this is not the end of the story, but rather, just the first step in making problems related to functional coupling more obvious"

conception, like in case of a merger or an acquisition. SOA and ESB offer ideology and tools that help integrate existing applications and facilitate future integrations. The primary theme of products that offer solutions to problems of high technical coupling is independence from technical peculiarities of the applications.

SOA/ESB products help overcome the following issues:

- ***Differences in transport protocols and message exchange patterns***: For example, one client may require synchronous invocation over HTTP, while another might require asynchronous calls over a reliable asynchronous transport accessed through JMS API. Today's popular solution is to use messaging products such as WebSphere MQ or Microsoft Message Queuing, and to use bridges to adapt applications that use other transports. WS-BPEL implementations (that provide orchestrating for Web services) are also used for translating message exchange patterns.
- ***Differences in data models***: For example, an EIS may use an internal format for purchase orders. If a decision to support a public standard like UBL is made, then an adapter service to translate incoming and outgoing messages between these two formats is required. SOA products provide extension points where such adapters can be added to supply auxiliary libraries and languages that will support format transformations.
- ***Legacy service integration***: SOA solutions allow interaction of different legacy applications based on various integration technologies. It might be supported by bridges between the technologies. For example, if a SOA application with SOA technical assumptions needs to interact with a CORBA application with respective assumptions, then a bridge translating SOAP messages to CORBA and vice versa can be used. Such integration is often almost transparent and changes in the legacy application sometimes can be avoided.
- ***Changes of a service location***: If an application accesses a service provided by another application, it should be ready for changes of the service location. This is usually achieved in one of the following two ways:
  - The application attempts to find the required service using a service directory mechanism such as UDDI. So if the service is relocated, the application can find it again in the new location.
  - The application uses a messaging product like WebSphere Message Broker. Here developers shift the responsibility for locating services to the messaging product which routes messages to correct destinations. Administrators have to update the product configuration if location of the service changes.
- ***Multiple authentication domains***: Sometimes different partitions of an EIS use different authentication mechanisms. In such cases, authentication information needs to be translated from one security domain to another. For example, WS-Trust and WS-SecureConversation implementations can be utilized.

These products reduce the amount of technical assumptions that have to be shared by applications, thus reducing technical coupling between them.

Although modern approaches are useful, we should not consider tools magic wands. They do not make developers free from knowing technologies and thinking about how to use them effectively. It is very easy to misuse the tools and to introduce security and performance problems, as described later in this article.

Another important point is that we cannot assume a migration to SOA being the last migration. SOA/ESB is based on experience gained from previous integration technologies like CORBA, and it will be superseded by technologies based on experience gained from SOA/ESB. This issue should be considered when the cost of migration to SOA is estimated. Early signs of coming transition are complexity and overabundance of standards.

When problems of technical coupling are solved, this is not the end of the story, but rather, just the first step in making problems related to functional coupling more obvious.

Functional Coupling

Functional coupling is the concept that can be applied to real business processes and paper-based bureaucracy. For example, if a member of an organization has to sign ten forms in different departments to access a document, then the workflow is tightly coupled with a number of other business processes in this organization.

In computer systems, high functional coupling may happen when highly coupled workflows are implemented because of pitfalls in functional requirements. In some cases computer-based implementation of business processes does not reflect actual processes (this might happen due to many reasons and in particular from restructuring an organization's workflow). In all cases, functional coupling is caused by what an application does rather than how. Therefore, such problems are solved by changing functional assumptions about business processes and implementing these changes.

Vladimir Shraibman is an analyst at Axmor. He has been working in the IT industry for more than 6 years as a programmer, software analyst and team leader. Vladimir was a member of Axmor's team prepared a series of articles devoted to migration of J2EE applications from JBoss AS to Geronimo AS and IBM WebSphere Community Edition.

shvb@isg.axmor.com

Functional coupling problems are often less obvious than the technical ones. They should be detected and addressed by analysts and domain experts who need to decide what requirements need modification.

An important aspect of activities that reduce functional coupling is reaching *functional cohesion*, meaning that application functionality should contribute to a single well defined task. UNIX commands and text utilities are classic examples of correlation between programs and specific actions.

Often application scope is extended on an ad hoc basis and the initial task gets lost in changes. It is like a library maintenance application that transforms into Internet bookshops by adding a few use cases here and there. Low cohesion of applications in an EIS leads to a number of problems. The most obvious are:

- Code is difficult to understand which raises expenses for each change.
- Applications are more highly linked and they are more likely to break if another one changes. Moreover, if these applications are maintained by different development teams, then making changes requires additional team coordination expenses.
- System failures happen more often, and the applications are difficult to test when business processes are spread among different applications and servers. This leads to increased development and testing efforts and increased bugs found after deployment.
- In systems with low cohesion, services are more often located on different servers. As a result the network load is raised, and more expensive hardware is typically required to achieve acceptable performance.

Sometimes functional coupling problems and a lack of functional cohesion can be noticeable to software architects. A possible indicator is absence of communication cohesion (i.e., when several different applications are responsible for operating the same business data). This makes an EIS more difficult to maintain and forces developers to deal with problems that could have been avoided if a more adequate domain analysis were done.

Requirements affect applications, but the reverse may also be true. An EIS that adequately implements business processes of an organization provides extremely valuable feedback about the organization's structure and workflow. Thus, if an application belonging to an EIS demonstrates poor cohesion, it may be a sign that underlying business processes could be improved to increase the degree of cohesion. Organizations should not neglect using this as a way of collecting helpful information.

Problems of high functional coupling cannot be reduced purely by technological means since the most difficult part is requirement gathering. The most that we can expect from tools is help in performing a re-factoring of an application and related workflows. Modern IDEs, such as JetBrains IDEA or Eclipse, provide great facilities for program source code re-factoring. We could also expect that in the future we will have tools to make changes on the level of Web services and business processes. For example, when the definition of a Web service is modified, an adapter Web service that provides older versions of the service could be automatically generated. This adapter can be helpful while migration to the new version of the service is in progress allowing updates within the EIS applications one-by-one. But in all cases, decisions about what refactoring procedures should be performed are taken on by humans.

Common Coupling Traps

Along the road to SOA, developers can easily increase coupling, while thinking they're decreasing it. Many coupling traps arise when developers make their systems conform to some additional requirements and principles by compromising its conformance to the original functional requirements. This often happens because IT departments are under increasing pressure to minimize the cost of changes, and developers believe that these additional principles will simplify modifications in the future.

However cost saving techniques that are successfully applied inside an application do not necessary work within the context of an EIS. This was discovered a long time ago (http://citeseer.ist.psu.edu/waldo94note.html), but many people continue learning it from their own experience. One of the important techniques that fail is

maximally generic, flexible, and fine-grained interfaces (the latter technique is sometimes over-applied even in the context of a single application).

Let's look at the most common practices that may increase cost of changes. Although the design patterns described below are acceptable approaches, in some cases, we recommend considering them as a warning sign.

Exposing the Application "As-Is"

There are a number of tools that simplify and automate design of Web services by generating a WSDL document from existing code. However, relying on existing internal interfaces can lead to the following problems:

- Analysis that was done during initial application development could be already obsolete at the time of WSDL development.
- Technical assumptions of existing internal components could be invalid when their functionality is exposed as a Web service. For example, an invoker could be specified as a user name parameter to method invocation. This solution could work nicely for local EJBs that are called from servlets, because it is assumed that servlets check user credentials. However if it is exposed as a Web service, it could create a huge security hole. Another similar example is when servlets are responsible for data validation and in these cases some business methods could rely on the servlets to validate data and skip checking it.
- Tools do not discriminate between functional and technical aspects of existing interfaces. They typically leak technical aspects of systems to ESB and often ignore existing standards for passing around such technical information. For example, client credentials could be provided in a custom manner instead of using WS-Security headers.
- Legacy services usually reflect technical approaches popular at the time of their design. If such services are translated by tools, they could be extremely inefficient.

The recommendation here is to design WSDL according to business requirements, ignoring technical aspects of previously implemented solutions. Technical aspects are better expressed independently of business functions. Development of a WSDL document should not be considered a problem, but an opportunity to specify or double-check the functionality being exposed.

Over-Generalized Interfaces

Another interesting trap is over-generalized interfaces. This trap has two popular incarnations: CRUD (Create-Read-Update-Delete) and vertical industry standards.

CRUD Web services describe generic operations over objects instead of specific business operations. CRUD interfaces are born out of the good intention to provide a single unified interface to all clients. CRUD operations also make it possible to generate a large part of the implementation skeleton using code generation tools.

To illustrate problems of CRUD interfaces, let's consider the case of a bank account. There could be operations to set account properties in a particular balance. A property might be updated only by an authorized request sender. While the CRUD approach looks very generic, it is difficult for implementation, use, and more importantly, security:

- On one hand, it is difficult to implement such a service. Security requirements are usually expressed for high-level operations. For example, some clients are permitted to deposit money, some to check an account, and others to withdraw. To check if a client is authorized to perform a low-level operation or update business data, the service has to guess what a high-level business action is and which is actually intended to be performed. Because authorization decisions are done inside methods, it is difficult to understand security policy without examining the operations implementation.
- On the other hand, the service is inconvenient for use by clients. They need to formulate high-level business operations in terms of low-level business data update operations. This also increases functional coupling between client and server, because the client has to know how the business operations affect business data. If the data schema is radically refactored, it will be very difficult to write a proxy for the old interface. It is likely that all clients will have to be updated.

The rule of thumb is that if a single operation does different things for different clients and decisions are made based on who the invoker is, this operation should be split into several independent operations.

Of course, sometimes CRUD interfaces are adequate. For example, they could be used for communication between JavaScript code executed on a client and server side logic. Here, tight coupling between these two parts of the solution is rather natural. However, using CRUD interfaces for Web services is often the wrong idea.

Vertical Industry Standards

A less radical variant of the previous trap is the use of a standard XML schema for business domain messages. Pertinent examples of specifications here are OASIS Universal Business Language and ACORD P&C. It may seem that such standards reduce coupling between systems, but

"There are a number of tools that simplify and automate design of Web services by generating a WSDL document from existing code"

> There is an opposite lure to make a system too coarse-grained, when several different business functions are associated, because they are implemented by one system"

actually, their effect on this aspect is not considerable or absent at all.

Often these standards declare a quite generic and flexible XML language that is able to accommodate different business requirements. However, the target service is usually able to understand only a subset of this rich language. Some constructs are valid for such a language and can be rejected by a Web service because they contain unsupported fields or miss required fields. UBL developers have noticed the problem and offered a partial solution in the form of customization guidelines.

If the schemata are used as is, implementation of both clients and servers is more complex. Servers are required to perform extra validation checks. Developers of clients are forced to check each step with additional guidelines to ensure that requests will be valid, instead of just relying on the respective API.

Although vertical industry standards can decrease documentation and interface development efforts, they often do little to reduce coupling of systems. Moreover, naive implementation of these standards can conceal existing coupling problems instead of solving them.

Fine-Grained Interfaces

There is a lure to make a system too fine-grained, i.e., more fine-grained than actual business operations. The primary motivation for this anti-pattern is to increase the flexibility of the system and to reduce the volume of the traffic. The system allows different invocation sequences, depending on what is required by clients, so that a client's costs are proportional to his needs.

This is a well-known bad practice that was understood as anti-pattern even during CORBA days. A typical example is expressing a high-level update operation as a sequence of property update operations. However, it still periodically surfaces because it falls into the exposing the application "as is" trap. The problem becomes a latency and per-request cost of network operations that consume more resources than local operations.

When a business function is split into too-small pieces, the coupling also becomes higher because the business operation becomes implemented by client applications rather than by the server logic. This also leads to security problems similar to ones already discussed above in over-generalized interfaces.

Coarse-Grained Interfaces

There is an opposite lure to make a system too coarse-grained, when several different business functions are associated, because they are implemented by one system.

ACORD P&C 1.x.x specification is an example of a specification in the domain of insurance. A single request can contain a number of business requests that create and update different business objects (claims, polices, etc). It is also possible to check the status of previous requests and pull results of completed operations. This is done to reduce the amount of interactions between clients and servers. Such an approach results in complex logic for request processing, and this complexity is caused not by the business domain of the specification, but by technical decisions to create a coarse-grained interface on the system and specify processing models for requests. As the result, the technical coupling in the system is increased since both clients and servers have to support the specified processing models.

Conclusion

RFC1925 says: "In protocol design, perfection has been reached not when there is nothing left to add, but when there is nothing left to take away" (http://ietf.org/rfc/rfc1925.txt). Humans naturally strive for perfection. However, in the rush of adding new application functionality and integrating existing applications, this desire often stays unrealized. And to reach perfection, it is required that managers, analysts, domain experts, and system architects closely and confidingly work together because each of these roles deals with only a piece of the whole puzzle.

Management is aware about the mission and the structure of their organization. Knowing needs, goals, and rules is the essential part of any change within an enterprise. Analysts and domain experts can elaborate these goals and rules to software requirements. Architects know how to bring these requirements to the life.

Coupling is an interesting and complex problem, which is created by decisions on many different levels. High coupling on the level of real business processes is often caused by management decisions. High functional coupling is often caused by the way requirements are distributed among applications, and it is mostly the responsibility of analysts and domain experts. High technological coupling is often caused by decisions made by architects.

While sometimes it is possible to partially compensate a problem of one level on other levels, it is still better to solve it on the level that it belongs to. RFC 1925 also says: "With sufficient thrust, pigs fly just fine. However, this is not necessarily a good idea. It is hard to be sure where they are going to land, and it could be dangerous sitting under them as they fly overhead."

There are tools and methodologies that help to reduce coupling on each level. However human intelligence and creativity are still the best and indispensable tools.

j-Interop: An Open Source Library for COM Interoperability Without JNI

A search for pure, non-native, bi-directional interoperability with COM servers

by Vikram Roopchand

I have spent a good part of the last year trying to "wrap" COM servers in Java for a content management organization. It had an array of *syndication* servers supported by an *integrated messaging* platform developed using COM. The purpose of this exercise was to increase the organization's market penetration by hooking on to the J2EE bandwagon across multiple platform configurations. With so many different complex COM servers to work with, some supporting *automation* and others not, I struggled with the all too familiar JNI cycle…code, crash, code some more, and then crash. Literally speaking, I must have brought down the JVM hundreds of times. To top it off, some syndication servers worked on a "pull" mechanism, they could pull the content out from the interfacing repositories. This meant bi-directional access and an event-based interoperation.

I had a look at some Open Source projects, but they were all using native libraries and didn't sufficiently meet the requirements in hand. Ultimately, we did complete a scaled-down version of the project. I'm quite sure that most of the requirements could have been handled if a non-native solution (in the Open Source community) existed to COM interoperability.

There are a number of reasons why I believe JNI shouldn't be used for accessing COM. It may work well for a small-sized project (two or three COM servers, no bi-directional access, etc.). But for any serious initiatives, I think it has certain drawbacks. Though JNI *is* Java, you still need to know a lot more about the native component and the target architecture before accessing its services. One has to be proficient in Java and in the native programming language as well (some Open Source projects are trying to address this complexity, i.e., trying to abstract JNI itself). With increasing demands to deliver on time – *before time*, and the delivery having quality, it's usually hard to find such cross-platform resources. The native and Java layers are so tightly coupled that having dedicated resources for each proves more of a headache than a solution.

I'm pretty sure that it's obvious to you that by linking with native code, JNI takes away one of the most powerful features of the language itself, "platform independence" (write once, run anywhere). It *binds* the application to the host platform. For example, when using JNI in a Windows environment, the Java application is forced to rely on the native DLLs for its functionality. The same application can't be ported without porting the native library to another platform. This also requires a *proxy/stub* approach when you want to access a COM server from Unix. This is an invasive procedure (you *are* deploying a potentially unknown code on the machine where the COM server is located, even if it's in the same intranet) and *may* not be allowed by the administrative policies governing the domain.

Another disadvantage, when linking with native code is the instability it may bring with it. A poorly written DLL (speaking of Windows) will bring down an entire JVM, taking with it some vital applications. And of course, try debugging that!
The architectures built on JNI look more or less like Figure 1.

Okay well…so what can be done about it? I sat down some months ago to develop *an open source library that implements the DCOM protocol, thus allowing for pure, non-native, bi-directional interoperability with COM servers.*

I'll try explaining my work in two steps. First I'll give you a primer on DCOM and its inner workings and then I'll talk about j-Interop. If you know how DCOM functions *under the hood*, you can skip to the section about j-Interop.

Vikram Roopchand is a Technical Architect working for Infosys Technologies Ltd. (www.infosys.com). He has about 8.5 years of experience and specializes in Cross Platform development across Content Management and Business Intelligence domains.

vikram_roopchand@infosys.com

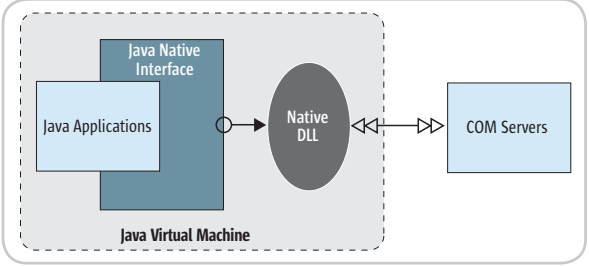


Figure 1

DCOM: What and How

Distributed COM or DCOM is a high-level network protocol developed to provide location transparency to COM-based components. The keywords being network protocol and location transparency. Traditional COM components can only perform inter-process communication across process boundaries on the same machine. DCOM uses an *RPC mechanism* to send and receive information transparently between COM components (clients and servers) on the same network. DCOM was first made available in 1995 with the initial release of Windows NT 4. Essentially, it serves the same purpose as CORBA or RMI. Please have a look at Figure 2.

In terms of an ISO OSI protocol stack, this is how it looks (see Figure 3).

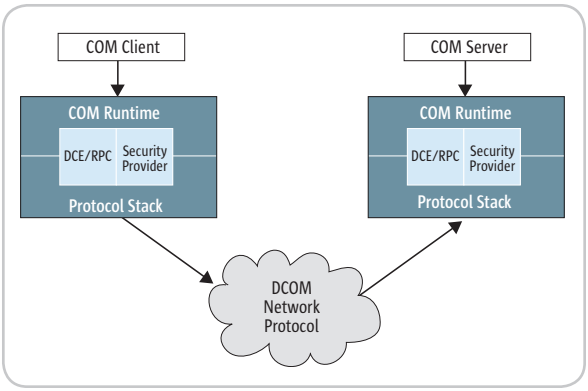


Figure 2

Application	DCOM
Presentation	NDR
Session	DCE/RPC
Transport	TCP
Network	Internet Protocol
Data Link	Network Driver
Physical	Ethernet Card

Figure 3 Windows TCP/IP based stack on Intel Platform

As you can see, DCOM is an application level protocol. It leverages most of the functionality offered by DCE/RPC (see the side bar for a brief overview of DCE/RPC). Since DCE/RPC isn't naturally object-oriented, Microsoft's implementation enhanced the protocol by adding new constructs and providing different meanings to some of the packet fields. This enhanced protocol was christened the Object RPC (ORPC) or MS-RPC. Another significant change that Microsoft made to the protocol was the addition of an NTLM Security Service Provider. One disadvantage of this was the impossibility of any interoperability with other implementations of DCOM on platforms that don't have NTLM, since NTLMSSP is available only in MS-RPC.

Let's see how DCOM works.

DCOM (MS-RPC): Under the Hood

The best way to explain the inner workings of DCOM would be by creating a (COM) client/(COM) server application. I've used Visual Studio to create one. Since "how" to create a COM server (from now on referred to as a COM component) is outside the scope of this article, I won't be mentioning it here. It's best left for a tutorial.

Our COM component is an "out of process" server, i.e., an EXE. For the sake of brevity, it has a single interface "ITestCOMServer" that has a single API call, "Add," that adds two integers and returns the result. Also note, this example is void of any error checks.

```
HRESULT Add (int x, int y, [out] int* result);
```

The COM client will execute this API.

The first step is to obtain the "handle" to the COM class serving this interface. The following calls instantiate the COM server and also provide a pointer to its IUnknown interface.

```
IUnknown *ptrUnknown = NULL;
IITestCOMServer *ptrTestServer = NULL;
HRESULT hr = CoCreateInstance (CLSID_ITestCOMServer, NULL,
CLSCTX_REMOTE_SERVER, IID_IUnknown, (void**)&ptrUnknown);
```

The second step entails getting a pointer to the actual interface in which the "Add" method resides.

```
hr = ptrUnknown->QueryInterface(IID_IITestCOMServer,
(void**)&ptrTestServer);

if (FAILED(hr))
{
cout << "Failed to get interface pointer, quitting";
}
```

The third step requires us to execute the "Add" API on the pointer obtained in step 2.

```
else
{
 int *result = new int;
 hr = ptrTestServer->Add(1,2,result);
 cout << "result of Add is " <<*result;
delete result;
}
```

The last step is to release all references to the COM server and let the COM runtime garbage collect it.

```
if (ptrTestServer)
{
ptrTestServer->Release();
}

ptrUnknown->Release();
```

Simple isn't it? Well, the COM runtime does a lot of work to orchestrate this cycle. Let me try to give you an overview of what happens behind the scenes. (Sources: www.opengroup.org, www.msj.com March 1998, DCOM specification)

1. Each Windows machine on the network has a subsystem known as the Service Control Manager (not to be confused with the Windows "Services" system). It's a DCE/RPC server that listens at port 135 and runs inside rpcss.exe. The SCM makes sure that when a client request is made, the appropriate COM server is connected and ready to receive the request. It provides an RPC interface, known as IRemoteActivation, which has only a single operation, "RemoteActivation," designed to activate a COM server on a remote machine. This, by the way, is an important difference between DCOM and classic RPC where the server must be running before the client can connect to it. The SCM resides at well-known endpoints, one for each supported network protocol (135 for TCP/UDP).
2. When the client gets a "CoCreateInstance" call with the execution context set as Remote (CLSCTX_REMOTE_SERVER specifies to the runtime that the COM server is located on a remote machine), the COM runtime consults the Windows registry for the "RemoteServerName" named-value. This value is located at [HKEY_CLASSES_ROOT\APPID\(CLSID of TestCOMServer)]. If a machine name is found under this

key then the request to activate the COM server is forwarded to SCM on that remote machine. The remote SCM uses the IRemoteActivation interface to activate the object identified by CLSID_ITestCOMServer.

3. What does it mean to "activate" a COM object? We will get to that after I talk about the IOxidResolver. I think it's important to clear these basics up otherwise things could become quite confusing.

   Each machine that supports the COM network protocol supports a one-per-machine service known as the "OXID Resolver." Like the SCM, it also contains an RPC interface "IOxidResolver." Oxid Resolver performs many important operations, primarily maintaining the binding information necessary to connect to the COM components being exported. It also takes care of keeping the exported objects alive by receiving pings from the COM clients (otherwise they'd be garbage collected) and does lazy protocol registration for servers scoped by the Oxid Resolver. I'll explain this last point a bit more. Each COM server can decide to support a certain set of protocols over which it can be contacted. For example, a server may want to answer only on UDP or TCP or HTTP or all three. Instead of reserving ports for each protocol even before it's activated, a server delays this to the time it's actually activated on a requested protocol. This is quite useful in preventing the machine from running out of ports.
4. Okay, coming back to activation. Activation should be seen as a set of activities that bring a COM server to a "ready to receive requests" state. In general, it means locating the COM server on the remote machine using the Windows registry, registering its connection information with the Oxid Resolver, marshaling the reference to its IUnknown (rather the "IRemUnknown") interface, and sending it back to the callee (explained in step 7).
5. On the remote machine, when the server is started by its SCM, two activities take place.
   - The server is associated with an "object exporter" and assigned an object exporter identifier (OXID). An object exporter keeps track of all the interfaces (like in our case ITestCOMServer), which this COM server will export or import.
   - The COM runtime also associates an "Oxid Object" with the COM server, which implements the COM interface "IRemUnknown." It forms the remote proxy for the base "IUnknown" interface. Please note the standard IUnknown interface is never remoted in COM. In its place, the IRemUnknown interface is remoted and results in local calls to QueryInterface, AddRef, and Release on the server.

     At activation time, the RPC binding information of the OXID is also registered with the server-side OXID Resolver. These are full bindings carrying the "How-To-Connect" information (including the supported protocol/port combination) of the COM server. This information is used by the underlying RPC mechanism of the client system to initiate a session with the COM server. One more point worth mentioning is that during activation the server has the choice of being "ready" now or waiting for the first call to come (lazy activation). Usually all servers prefer to be lazy till an actual call comes (initially IRemUnknown) on a specific binding.
6. I've talked about the SCM and the Oxid Resolver service. They are important infrastructure services. One provides for activation and the other for discovering the path and means to the activated object.
7. Up until now, in executing "CoCreateInstance" we've been able to activate a COM object. We still need to return the interface pointer, which will uniquely identify this activated COM object and allow us further operations on it (like a QueryInterface). Microsoft extended the Network Data Representation (the presentation layer protocol responsible for packaging semantics of the DCE/RPC datatypes) to add the concept of a "Marshaled Interface Pointer" (MIP from hereon). The MIP symbolically represents an interface reference to an object. It consists of two elements, an array of bytes and a marker specifying how to interpret this array of bytes. There are three variations to the interpretation, but I'll stick to the STANDARD type.

   The array of bytes representing the STANDARD interface pointer consists primarily of a 128-bit GUID known as IPID, short for interface identifier, that uniquely identifies an interface – it has a one-to-one mapping with each marshaled interface, i.e., ITestCOMServer will have a single IPID – an OXID and a 64-bit object ID (OID) that uniquely identifies the object on which the IPID is found. There's one-to-one mapping between an object instance (implementing one or more interfaces thus IPIDs) and the OID. This OID is quite useful during pinging. Along with all this the MIP also contains the full bindings for the OXID Resolver service running on the remote machine.
8. When the marshaled interface pointer is returned to the client side through the server-side and client-side SCMs, the COM runtime extracts the OXID, addresses the remote OXID Resolver from MIP, and calls the ResolveOxid() method on its local OXID Resolver to get the bindings ("how to connect" information) identified by the OXID (it has to reach the COM server now for further operations).
9. The clients-side OXID Resolver checks to se if it has a cached mapping for the OXID; if not, it invokes the ResolveOxid() method of the server-side OXID Resolver – it can since it has the address information from the MIP – which returns the registered RPC binding of the COM server.
10. The client-side Resolver caches the mappings, and returns the RPC bindings of the COM server to the COM runtime. This lets the runtime create an RPC channel that's connected to the Object exporter of the COM server.
11. The CoCreateInstance call is now complete.

Phew! I have drawn a picture to explain this entire cycle, just follow the alphabet.

I hope I was able to provide a concise introduction to the discovery and activation cycle. Let's see how the "QueryInterface" works and how the API call "Add (…)" is handled.

12. The remote connection has now been established and we have an interface pointer to the remote IUnknown (the "IRemUnknown"). A QueryInterface () call to obtain the "ITestCOMServer"" interface will result in the following activities taking place.

13. When the client invokes the IUnknown::QueryInterface, the COM runtime invokes the IRemUnknown::RemQueryInterface method on the OXID object in the target object exporter. The OXID object then invokes the QueryInterface() method on (possibly multiple) interfaces in the exporter (remember, the exporter knows all the interfaces the COM server has exported out). Please note that the IRemUnknown::RemQueryInterface method differs from the IUnknown::QueryInterface method since it can request several interface pointers in one call. The standard IUnknown::QueryInterface method is actually used to carry out this request on the server side.
14. If found, the requested interface from the COM server is then marshaled as a MIP and sent back to the callee. This MIP carries a new IPID symbolizing this interface. The OXID and OID remain the same.
15. Now that we have the interface pointer let's call "Add" on it. Upon receiving the ptrTestServer->Add(1, 2, int *) call, the COM runtime marshals the parameters in the NDR format and channels the request to the target object exporter identified by the OXID-resolved RPC binding (we did that in steps 9 and 10).
16. The COM runtime on the server side finds the target interface based on the IPID that's contained in the RPC header. This IPID is of the ITestCOMServer interface that was returned to the client during a previous QueryInterface call.
17. With the help of the Object exporter, the COM runtime invokes the method Add(…) on the correct interface of the COM server object and marshals the return values in the NDR format. These values are sent back to the callee using RPC infrastructure.
18. The same cycle follows during the Release() call, the COM runtime invokes the IRemUnknown::RemRelease() method on the OXID object in the target object exporter. The OXID object then invokes the Release() method on (possibly multiple) interfaces in the exporter.

This completes the entire cycle of obtaining an interface, executing an API on it, and subsequently releasing it. There's an additional task that the COM runtime also does and that's keeping the COM Server objects alive during a session. This is done via a ping mechanism. Pinging is carried out on a per-object (per-OID), not a per-interface (per-IPID) basis. Architecturally, at its server machine, each exported object (each exported OID) has associated with it a "ping period" that must elapse without getting a ping on that OID before all the remote references to IPIDs associated with that OID can be considered to have expired. Once expiration has occurred, the interfaces behind the IPIDs might get reclaimed (exactly when is implementation-specific).

I've tried to keep the explanation of DCOM internals as simple as possible. There's much more that happens, but I'm running out of space and I think you're running out of patience, so let's get to j-Interop.

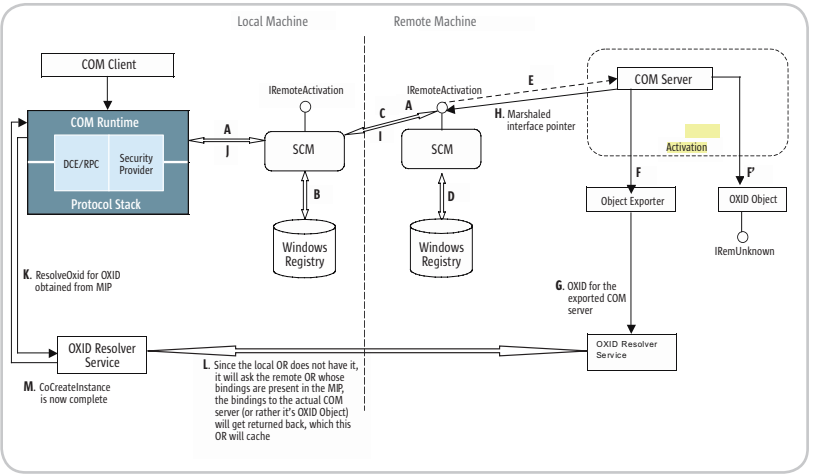


Figure 4

j-Interop

Suffice it to say j-Interop implements almost the entire DCOM protocol with its own Oxid Resolver service, pinging mechanism, Object exporter, etc. These are required for handling event callbacks, proxying a Java server in place of a COM server (bi-directional access), and making sure that the COM server isn't garbage collected while a client is connected to it and vice versa.

The library comes with pre-implemented packages for automation. This includes support for IDispatch, ITypeInfo, and ITypeLib. For more flexibility, it provides an API set to invoke operations directly on a COM server without going through automation. Another important feature is to allow full access and manipulation of the Windows Registry in a platform-independent manner.

The implementation has been tested on all advanced Windows and Fedora platforms and displays upward compatibility from JRE 1.3.1. For more technical specifications please visit http://j-interop.sourceforge.net.

I'll show you an implementation using j-Interop to call the ITestCOMServer from Java is Listing 1.

As you can see it's pretty straightforward. The ITestCOMServer supports the IDispatch interface. I've shown both ways of accessing the COM server, i.e., via the dispatch interface as well as via a direct call. From my experience, I'd suggest using the IDispatch interface, whenever it's available. It's much easier to program that way.

DCE/RPC

DCE/RPC stands for Distributed Computing Environment/Remote Procedure Calls. It originated from Network Computing System (NCS) RPC developed by Apollo (which later got acquired by HP).DCE/RPC specifies a complete set of APIs and Models to abstract the usual nuances of an RPC system like a named lookup, subsequent handshake\binding, passing of call data between two parties , handling communication errors, security etc.. It provides protocol support for both Connectionless and Connection Oriented communication and has a wide transport base, UDP, TCP/IP, SMB, HTTP to name a few. It also has a generic security model supporting several authentication mechanisms such as DCE, Kerberos and GSSAPI.

The full specification can be obtained from www.theopengroup.com.

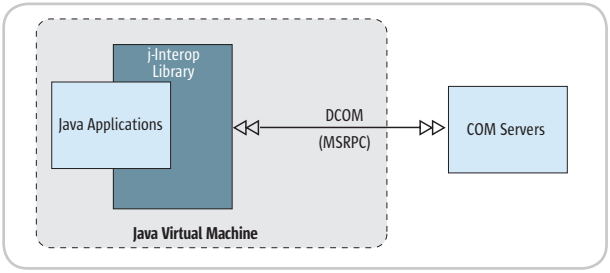


Figure 5

Figure 5 is normally how a Windows COM client communicates with its COM server. j-Interop does this as well and so for the COM server it's like an ordinary COM client. Whether the Java application is on Windows or Unix, it doesn't matter.

More examples and documentation can be downloaded from the SourceForge site mentioned previously.

The advantages offered by using a non-native library like j-Interop include:

- Clean integration of two of the leading technologies without writing any native code: j-Interop eliminates any need to write native (JNI) DLLs, cutting development time, and shortening the entire software lifecycle for the products (based on j-Interop). Such products are also saved from any kind of instable functions that result from poorly written native code (DLLs).
- Accessing COM components from any type of Java client, including applets, EJBs, servlets, JSPs, and standalone applications: Since it's pure Java, j-Interop can be used within *any* J2EE server and on *any* platform (that supports Java).
- Maximizing reuse of existing Java and COM components: All the plumbing on interoperating with COM servers is done by j-Interop, it makes *reusing* same components again, instead of porting back and forth between domains, a more *lucrative* and *viable* option.
- There should no longer be a dependency on cross-platform resources thus minimizing cost and improving quality: The same resources for Java can be used without any additional training/competency building and the turnaround time can be brought down substantially. Not having to deal with native code also removes the complexity associated with maintaining the same. The code is now much cleaner and distinctly segregated between the two domains. Debugging the projects based on j-Interop is substantially easier than debugging JNI-based DLL projects.
- Easier deployment since there's no custom code at the server: No special treatment has to be given to j-Interop clients, they just behave like standard DCOM clients. This is a big advantage in terms of administering the machines where the COM servers are deployed. The administrator doesn't have to care about security or the instability of native components bringing the server down (Denial of Service).

Listing 1

```
 public TestCOMServer(String address, String[] args) throws
JIException, UnknownHostException
{\
JISession session = JISession.createSession(args[1],args[2],args[
3]);

   //instead of this the ProgID "TestCOMServer.ITestCOMServer" can
be used as well.
   //comStub = new JIComServer(JIProgId.valueOf(session,"TestCOMSer
ver.ITestCOMServer"),address,session);
   //CLSID of ITestCOMServer
   comStub = new JIComServer(JIClsid.valueOf("44A9CD09-0D9B-4FD2-
9B8A-0151F2E0CAD1"),address,session);
}

public void execute() throws JIException
{
   unknown = comStub.createInstance();
   //CLSID of IITestCOMServer
   IJIComObject comObject = (IJIComObject)unknown.
queryInterface("4AE62432-FD04-4BF9-B8AC-56AA12A47FF9");
   dispatch = (IJIDispatch)ComFactory.createCOMInstance(ComFactory.
IID_IDispatch,comObject);

   //Now call via automation
   Object results[] = dispatch.callMethodA("Add",new Object[]{new
Integer(1), new Integer(2), new JIVariant(0,true)});
   System.out.println(results[1]);

   //now without automation
   JICallObject callObject = new JICallObject(comObject.getIpid());
   callObject.setOpnum(1);//obtained from the IDL or TypeLib.
   callObject.addInParamAsInt(1,JIFlags.FLAG_NULL); //setup the
first int param
   callObject.addInParamAsInt(2,JIFlags.FLAG_NULL); //setup the
second int param
   callObject.addInParamAsPointer(new JIPointer(new
Integer(0)),JIFlags.FLAG_NULL);
   //Since the retval is a top level pointer , it will get
replaced with it's base type.
   callObject.addOutParamAsObject(Integer.class,JIFlags.FLAG_NULL);

results = comObject.call(callObject);
   System.out.println(results[0]);
}
```

ALL ATTENDEES RECEIVE: Certificate of Completion PLUS Course Materials on DVD!*

AJAX ONE-DAY HANDS-ON INTENSE TRAINING!

AJAXWorld University's **"AJAX Developer Bootcamp"** is an intensive, one-day hands-on training program that will teach Web developers and designers how to build high-quality AJAX applications from beginning to end. The AJAX Developer Bootcamp is intended to be the premier AJAX instructional program presently available anywhere.

AJAXWORLD UNIVERSITY BOOTCAMP

www.AJAXBOOTCAMP.sys-con.com

Roosevelt Hotel
New York, NY
January 22, 2007

Roosevelt Hotel
New York, NY
March 18, 2007

What You Will Learn...

Overview of AJAX Technologies
– HTML vs. DHTML
– Network Concerns
– Asynchronous Conversations with Web servers
– The characteristics of high-quality AJAX applications
The Web page is the application
What the server provides
User interaction

Understanding AJAX through the basics of AJAX
– Asynchronous server communication
– Dynamic HTML,
– Javascript Design patterns
– User interface strategies for building elegant, highly addictive Web sites and applications
– The Essential AJAX Pieces
Javascript
Cascading Style Sheet (CSS)
Document Object Model (DOM)
XMLHttpRequestObject
– The AJAX Application with Javascript
– Using CSS
– Structuring the View Using the DOM
Applying Styles with Javascript
Communicating with the Web Server in the Background
Designing AJAX Applications
Design Patterns
– Introduction to AJAX Frameworks-
Dojo, script.aculo.us, Prototype
Over of framework capabilities
– Examples of frameworks in use
– Best Practices

Hand-On Development The Fundamentals: Building the Framing for an Ajax Application
– Review the courseware code with the Instructor
– Begin building a working AJAX application and start applying technique and technologies as introduced in class
– Create the basic AJAX application by creating HTML, Javascript, and CSS files
– Learn Best Practices and Validation
– Learn and add script.aculo.us effects
– Learn and add the Dojo Framework

Adding Basic Ajax Capabilities to a Web Page: Going Deep Into the AJAX User Experience
– Elements on the Rich Internet Experience
Interactivity
Robustness
Simplicity
Recognizable Metaphors
Preservation of the Browser Model Bookmarks/Back Button
– Background operations
– Building a AJAX Notification Framework
– Provenance and Relevance
– Rich Experience Support with Third-Party AJAX Client – Framework
– Using AJAX layouts, containers, and widgets
– Patterns for Animation and Highlighting
– User Productivity Techniques
– Tracking Outstanding Network Requests

Hands-On Development: Expand the Application with more Advanced Ajax
– Review the courseware code with the Instructor
– Expand the Mural Application
– Add Features using Dojo
– Add specific Dojo Libraries to support Ajax widgets
– Work on server side communications in the background
– Create a tabbed layout
– Create a submission form to upload to the server, all without reloading the page
– Create an Ajax submission form that will take uploads on one tab.
– Created a form validation that ensures only the right information is submitted.
– More Stretch work for those who want to learn additional concepts

Advanced AJAX Concepts
– Review Ajax Concepts
– SOA and Mashups
– Current state of Ajax Frameworks
– Web 2.0 and the Global SOA
– Ajax Constraints
– Design Patterns
– Javascript Timers
– Ajax Programming Patterns
– Performance and Throttling

Hands-On Development: Working with Advanced Ajax Capabilities
– Review the courseware code with the Instructor
– Work with the Accordion control
– Learn how to use the Tree control
– Explore Dojo's animation capabilities
– Explore how the debug output can be used in elements
– Tour Dojo and RICO demos
– Experiment with new Dojo features from the Dojo demos source code and attempt to add them to various parts of the Mural application
– Overview of Future of AJAX and Rich Internet Applications

*ALL RELEVANT COURSE MATERIALS WILL BE OFFERED ON DVD AFTER THE EVENT

HURRY! REGISTER NOW FOR EARLY-BIRD DISCOUNT!

What Attendees Are Saying...

"The trainer was excellent. The material too!"

"The hands-on, although long, was useful and educational!"

"The instructor was good. He answered questions thoroughly!"

"Well designed and organized. Good mix of lecture vs lots of hands-on!"

SYS-CON EVENTS For more great events visit www.EVENTS.SYS-CON.com

Joe Winchester
Desktop Java Editor

Ten Brilliant Years

The year 2006 marked the tenth anniversary of the Java language and for me is the most significant in its history.

The most important event was the announcement that a GPL version of Java SE will be available sometime in the first half of 2007. If nothing else, all the back and forth "will they, won't they" discussions over open source have been a distraction for the Java community. They also provided a source of FUD to those who don't believe in Java, enabling them to describe the community as divided, fragmented, and imploding under its own mass of internal fighting. I don't believe for a second that any of this was actually occurring; however, some customers I spoke with did have this perception of divided community. Far from it, the Java community is an incredibly healthy place where the pace of innovation and ability to adapt occurs faster than in any other technology space. The ingredients for this are the mixture of mom and pop teams who create elegant and nimble frameworks that become overnight de facto ways to do validation, navigation, or persistence, while working hand in hand with large organizations whose stock value is based on reliability, serviceability, and portability of the language. Every JavaOne question and answer session I've attended over the years invariably had someone in the audience standing up and berating an onstage developer for a particular bug that hadn't been fixed for the last *n* years. The answer was always one of prioritization and that the development team had more line items than they could accomplish with the available resources. For the questioner it's an answer akin to, "Your top problem didn't make our top 500." Now the reply can be, "Would you like to be a committer? Would you like to help us do some testing with our release so we can verify your patch?" It's welcoming, it's inclusive, it's how to move things forward, and for me it's the fuel for the feedback loop that makes open source community projects become better at a rate that equals the number of smart, willing, and motivated users.

The second most significant event for me in 2006 was the announcement of the Google Widget Toolkit (GWT) (http://code.google.com/webtoolkit/). It's a brilliant piece of work designed by some very talented people at Google. From a solution point of view, GWT allows developers to write Java code that can be deployed in a browser and achieve the kind of dynamic Web 2.0 functionality that all the Web heads get excited about.

Under the covers it does this by compiling the Java to be deployed as a mixture of HTML and JavaScript. What's exciting about GWT isn't just that it's a very cool piece of technology, but also the concept behind how it is using the Java language. Java's founding mantra is "write once, run anywhere." For most of us this translates into "compile to bytecodes and run on a JVM that abstracts the operating system." This doesn't always meet the scenario, however, as evidenced by something like Java applets that are no longer relevant to all but a few die-hard Web page developers. In their place the "cool effects" brigade resort to stuff like AJAX, Flash, and other technology that, while optimized for browser deployment, are certainly not optimized for development. Watching an AJAX developer is rather like watching a C coder of yore struggle with primitive tools and obtuse syntax. Java applets failed because they treated the browser as a delivery mechanism for .class files to the desktop that needed to have a compatible JRE. What if instead you regarded the browser as a smarter beast and used its APIs as a virtual machine you could run within? This is the magic of GWT: it takes the beauty of the Java language with its plethora of high-level development tools and programming suites, then compiles this to HTML and JavaScript. Java has now become a fourth-generation language with the browser being the runtime.

The third most significant event for me in 2006 goes jointly to Eclipse and NetBeans.

Eclipse celebrated its fifth birthday as an open source project, and it's one that has gone from strength to strength each year. I've been fortunate to have been involved with Eclipse from the outset and the thing that pleases me most each year at their annual EclipseCon conference is how the buzz and excitement moves and changes around. One year Web tools are the hot topic, the next year it's the rich client platform. Not only does the technology's focus shift, but the people do too, as new companies and new stars shape and drive its future.

NetBeans is often seen by some as a rival to Eclipse and vice versa, viewpoints I used to hold myself. I regard them differently now, with NetBeans holding the battle standard for Java, giving it a sweet-tasting all important out-of-the-box first kiss experience, a platform that keeps pace with the latest JSRs and language features so they are showcased in IDE samples and tooling rather than PDF specifications; a tool is to Java what VisualStudio is to the Microsoft runtimes. For Java to remain relevant and grow in the next 10 years, we have to look at those companies in whose interests it is to see us fail, work out what makes them successful, and compete with them on their own fronts. The key battles will be fought in ease of use, growth and adoption by customers who feel confident and secure in its future, and adaptability to new scenarios. Java's tenth year laid down some very firm roots to enable it to compete in all of these spaces, and I hope that the next 10 bear fruit and see the language go from strength to strength.

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

joewinchester@sys-con.com

REGISTER TODAY AND SAVE!

Rich Internet Applications: AJAX, Flash, Web 2.0 and Beyond...

www.AjaxWorldExpo.com

AJAXWORLD™ EAST
CONFERENCE & EXPO

NEW YORK CITY

THE ROOSEVELT HOTEL LOCATED AT MADISON & 45th

SYS-CON Events is proud to announce the AjaxWorld East Conference 2007!

The world-beating Conference program will provide developers and IT managers alike with comprehensive information and insight into the biggest paradigm shift in website design, development, and deployment since the invention of the World Wide Web itself a decade ago.

The terms on everyone's lips this year include "AJAX," "Web 2.0" and "Rich Internet Applications." All of these themes play an integral role at AjaxWorld. So, anyone involved with business-critical web applications that recognize the importance of the user experience needs to attend this unique, timely conference – especially the web designers and developers building those experiences, and those who manage them.

BEING HELD MARCH 19 - 21, 2007!

We are interested in receiving original speaking proposals for this event from i-Technology professionals. Speakers will be chosen from the co-existing worlds of both commercial software and open source. Delegates will be interested in learning about a wide range of RIA topics that can help them achieve business value.

Where's *i*-Technology Headed in 2007

SYS-CON Media's Annual Poll of Industry Prognosticators

by Jeremy Geelan

At the end of each year, when SYS-CON informally polls its globe-girdling network of software developers, industry executives, commentators, investors, writers, and editors, our question is always the same: where's the industry going next year?

Every time, the answers are surprisingly different from the year before, and of course throw light not just on where the industry is going but also how it's going to get there, why, because of who, within what kind of time-scale – all that good stuff.

Enjoy!

Ruby on Rails • JRuby • AJAX • Rules-Based Programming

JASON BELL
Enterprise Developer, Editorial Board Member, *Java Developer's Journal*

My predictions for 2006....

1. ***Incremental mainstream adoption of Ruby on Rails***
   It's going to happen, isn't it? Keep an eye out for Sun's offering of JRuby. Whether this is the death of other open source scripting languages like Groovy remains to be seen. Ruby has been a wake-up call and has now drawn the line dividing serious scripting languages from "hobby" languages (ones that wouldn't see enterprise adoption). For me, my job just got a whole lot easier, a whole lot quicker.

2. ***A slowdown in the AJAX hype***
   I think the shine has worn off. There are some nice applications about but at the end of the day it's a Web page with some very fancy JavaScript.

3. ***2007 is the year of the business rule***
   Rules-based programming will be big business. With the likes of JBoss acquiring Drools it's certainly an area to keep an eye on.

LAMP • REST • ATOM • Apple

DAVID HEINEMEIER HANSSON
Creator of *(Ruby on) Rails*

1. 2007 will be the year where LAMPers finally decide to stop being neutral about the WS-* mess and pick the side of REST: the next wave of Web APIs will stop supplying both a SOAP and REST API and just go with the latter.

2. On the leading edge, we'll see the same for RSS vs ATOM. For techies in the know, ATOM will become the assumed default syndication format and that'll mark the slow decline of RSS (though more as a technology than as a brand, RSS will remain synonymous with feeds).

3. Apple will continue to trounce everyone else for the preferred geek platform. The stigma of being a Web programmer still using Windows will increase.

Vista • Office 2007 • Zune • AJAX • Ruby • Java Ruby on Rails • Flash Memory

GARY CORNELL
Founder & Publisher, *Apress*

In no particular order:

1. IE 7 will have a fast adoption curve and so Firefox will cease gaining market share.
2. Vista will have a slow adoption curve.
3. Office 2007 will have a slower adoption curve.
4. Oh, the Zune will have no adoption curve.
5. The AJAX bandwagon will gain even more speed.
6. Ruby's momentum will slow down as Python and PHP frameworks to combat Rails grow in popularity.
7. The open-sourcing of Java will have no effect whatsoever on Java's slow decline in favor of dynamic languages (Ruby, Python) and C#.
8. Sales of high powered desktop will slow.
9. Apple will no longer gain market share for its desktops and will stabilize at its current meaningless level.
10. Ultra lightweight notebooks based on flash memory with instant on/off will start coming out in large numbers.

SOA & Web 2.0 • "Outside-In SOA" • Semantic Web • AJAX

DAVID S. LINTHICUM
CEO, *The Linthicum Group*

1. ***The worlds of SOA and the Web 2.0 blur together.*** While many who think SOA don't think Web 2.0, and many who think Web 2.0 don't think SOA, those days will come to a fast end in 2007. So, what does this mean to those standing up SOAs today? It's clear that many of the services we consume and mange going forward will be services that exist outside of the enterprise, such as subscription services from guys like Salesforce.com, or perhaps emerging Web services marketplaces from guys like StrikeIron, Google, Amazon, and others. This is outside-in SOA, in essence reusing a service in an enterprise not created by that enterprise, much as we do today with information on the Web. Thus, those services outside of the enterprise existing on the Internet create a Universal SOA, ready to connect to your enterprise SOA, perhaps providing more value.

2. ***The rise of the Semantic Web.*** The Semantic Web is the abstract representation of data on the World Wide Web, based on the Resource Description Framework (RDF) standards and other standards. Although this notion has been around for some time, in 2007 it will greatly affect how we design, build, and deploy Web 2.0 applications and SOAs, providing a mechanism to track and leverage application semantics, local and remote.

3. ***Enterprise applications continue to move outside the enterprise.*** With the success of Salesforce.com and many others, we'll continue to see applications move to the Web including accounting, CRM, HR management, logistics, inventory management, etc. While many Global 2000 companies will fight this trend, the success of the younger and more nimble up-starts will drive this movement quickly.

4. ***The success of AJAX drives traditional software back to the drawing boards.*** With the ability to finally provide dynamic rich content and applications over the Web, traditional software vendors will find that they need new products to play in this new world. Indeed, as Google Mail is giving Microsoft fits, so will other more innovative Web-delivered applications leveraging rich client technology such as AJAX. Entire interfaces will have to be rewritten to support AJAX, and end users will demand that we move away from traditional pump-and-pull HTTP programming.

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

jeremy@sys-con.com

Mobile AJAX • "Mobile Web 2.0" • SMS • LBS Flash Lite • On-Device Portals

LUCA PASSANI
Wireless Guru & Technology Evangelist, *Openwave*

Here are my predictions for 2007:

1. AJAX will still be hyped, but we will still see no mobile AJAX-based killer apps, only proofs of concept.

2. JAVA ME will not gain much more ground. Too fragmented. Games and some other apps. No killer apps though.

3. What people call "Mobile Web 2.0" is not Web browsing. Saying that mobile and Web will converge is trendy in some environments these days. This is wrong and that's hardly surprising: people buy phones to make calls, not to browse the Web, so why should we expect phones to get so much better at browsing the Web?

4. SMS will still represent 80% or more of data traffic. The rest will be downloads: ringtones, wallpapers and games. WAP will be mostly used as a discovery mechanism to get to those contents. Reformatting proxies to adapt Web content for mobile will be implemented by most operators. They will increase browsing a bit, but nothing earth-shattering.

5. Not sure about Location-Based Services. LBS have been on the verge of explosion for some time now.

6. Flash Lite will make significant progress in Europe and North America, also on operator portals.

7. On-Device Portals are an interesting development: content gets pushed to devices while the user isn't watching and they may decide later to buy it or not. This will be trendy next here. It will be interesting to see which actual implementations of the concept deliver.

8. More people will realize that device fragmentation is one of the main hurdles for mobile.

Flash Memory • AJAX Productivity • Red Hat • Vista Notebooks • Ubuntu

MARK HINKLE
Editor-in-Chief, *Enterprise Open Source Magazine*

Here are my predictions:

1. ***Flash-bootable PCs*** – It's been a long-time coming but laptop PCs will start booting from flash memory. This will make a huge difference in battery life. Intel will lead the way pushing their NAND flash boot memory on a new laptop platform and Apple will be among the first to adopt. The One Laptop per Child initiative will also provide a demonstration of the first zero disk drive PCs albeit small. Devices like this will inspire creativity on higher end models especially as the price of non-volatile memory continues to drop.

Open Source Java • General Public License v2 • GPL v3

Consequences of Open-Sourcing Java
by Tony Wasserman
Professor of Software Engineering Practice at Carnegie Mellon West and Executive Director of the Center for Open Source Investigation (COSI)

The open-sourcing of Java under the GPL 2 license will have a ripple effect on various technical and business issues in 2007:

- First, people will closely study the Java source code and find one or more serious bugs, at least one of which has been there since the earliest days of Java.
- Second, a real-time systems vendor will fork the source code, as permitted by the GPL, and create a variant that is "tuned" for real-time applications. This step will be the focus of a major debate within the Java community.
- Third, the open-sourcing of Java will have a positive impact on the adoption and use of open source software in general.
- Fourth, the use of the GPL 2 for open-sourcing Java will inhibit the completion and acceptance of the GPL 3 proposal.

2. ***New Crop of AJAX Productivity Applications*** – While the buzz around AJAX may fade, the number of robust new AJAX-enabled applications will increase. These applications will be built on evolving AJAX frameworks like Dojo and Rico and commercially backed platforms like OpenLazlo. Of course every new start-up will be secretly hoping for Google to make a bid and join the family that has been expanded this year by Writely and Jotspot.

3. ***Red Hat Will Become an Acquisition Target*** – Someone will make a bid on the #1 Linux vendor. Maybe Oracle who has done a number on the leading Linux vendor with Unbreakable Linux will take advantage of Red Hat's near 52-week low. Uncertainty and ambiguity in the enterprise Linux market will send Red Hat looking for another partner to avoid being swallowed by the DB maker. Maybe IBM will become Red Hat's white knight.

4. ***Open Source Everywhere*** – More and more companies will open source legacy products and launch new ones under open source licenses. Database vendor Ingres is going to set the standard that other more conservative infrastructure vendors will follow. Look for new open source initiatives from major infrastructure vendors like BMC, VMware, and even Microsoft.

5. ***Microsoft Vista Launch Will Boost Sales of Other OSes*** – Microsoft's launch of Vista will start to prompt hardware refreshes which can be nothing but good for Apple. Apple already has momentum, Intel hardware, dropping prices and all the tumblers are becoming aligned for it to creep above its measly 5% market share. Linux desktop vendors will likely see a few defectors from the Redmond camp, though big ships turn slowly. Look for Ubuntu to be the Linux desktop distribution of choice.

6. ***Half of All New PCs Will Be Notebooks*** – PC buyers are buying more notebooks every quarter and sometime in 2007 the number of shipping notebooks will match the number of desktop PCs or come very close.

IT Enabled Services • Web TV • Visual AJAX IDE Microsoft Atlas • Apache XAP

COACH WEI
Founder, Chairman, & CTO, *NexaWeb*

Here are my submissions:

1. ***IT Enabled Services is going to fly high in 2007.*** As a result, we will see:
   a. A lot more venture capital investments into IT Enabled Services;
   b. Of course, a lot of startup activities in IT Enabled Services (new company creation, merger and acquisition);
   c. There will be some significant moves made by "traditional, big companies" into IT Enabled Services too. For example, some of the possibilities are:
      i. Massive reality shows on the Web, instead of being on TV. Can you imagine "American Idol" on the Web? Speaking of this, I think highly of Yahoo's initiative into this area, including its recent acquisition of Bix.
      ii. A major entertainment company (NBC, ABC, etc.) fully embracing Web TV.

2. ***AJAX grows up*** – which means the following are available and useable:
   a. ***Visual AJAX IDE*** (solving the ease-of-development issue. Most likely based on Eclipse ATF);
   b. ***Declarative AJAX Framework*** (solving the ease-of-development issue. Most likely based on Microsoft Atlas and Apache XAP);
   c. Adoption of AJAX within less leading-edge enterprises.
   d. AJAXWorld Conference overtakes JavaOne conference. JavaOne is being renamed as JavaScriptOne Conference.

3. Growing adoption of Web 2.0 technologies within the enterprise
   a. Enterprise Mashup Server emerges as a product category.
   b. Less leading-edge companies start to adoption Web 2.0 technologies.

4. The IPO market shows signs of opening up
   a. One or two Web 2.0 companies go public, the majority of the exits are acquisitions.
   b. An increase of IPO filings and going public.

WS-BPEL 2.0 • BPM & Web 2.0 • SOA • XSLT • JSON

JOHN EVDEMON
Architect, *Microsoft*, with the Architecture Strategy Team focusing on BPM and SOA

E.F. Schumacher, a well-known British economist, once wrote: "I cannot predict the wind but I can have my sail ready." With that thought in mind here are ten predictions and hopes to help get your sails ready for 2007:

1. ***The WS-BPEL 2.0 specification will finally be approved as an OASIS standard.*** Adoption of WS-BPEL will initially be slow, driven by customer demand. BPEL will evolve beyond a "check box requirement" if people begin using it as a foundation for defining process profiles (conceptually similar to how people use WS-Security today). An updated mapping from BPMN to WS-BPEL will also be published.

SaaS • Open Source • Mobile • Enterprise IM Social Networks • China • Virtualization • RFID • Internet Video Prediction Networks • Intelligent Wikis

The Z Generation CIO & IT Professional
By Bob Zurek
Director of Advanced Technologies with IBM Information Integration Solutions

The Z Generation is typically described as those who are born sometime during the early 2000s and continue to the 2017 time frame. So what will these Generation Z IT Professionals be experienced with. Here's my prediction for the Top Ten characteristics (and this is just the tip of the iceberg):

1. Grew up in the world of SaaS and open source and wonders why you would ever license and install software. If you still needed to install software, then it should be available in the form of open source. Expects all internal projects to be developed using the open source model.
2. Grew up with a mobile technology and wonders how anyone could run a business without it. Insists everything be available on a highly portable digital device and everyone in the organization have a device. No exceptions.
3. Grew up knowing how to leverage the power of social networks for the benefit of the corporation. This includes the ability to build out these networks and use them to help build new products and technologies. Generation Z CIOs will have a huge advantage as they have grown up as participants in many social networks. China will be a big source of these networks. Websites will be built by the Z Generation CIO to invite outsiders in to help build new and innovative products that have yet to be thought of by the enterprises internal employees.
4. Grew up using Instant Messaging and will insist that the enterprise use IM as a priority over email and that email will only be used if the communications can't be done using the features of the future enterprise IM platform.
5. Will tap into offerings such as TopCoder.com to supplement project teams. There will be a world of competing Topcoder.com like sites where the best coders in the world will be found to solve very complex algorithms and other challenging software projects facing the IT department. China will be a major provider of these teams.
6. Grew up with a complete understanding of the value of virtualization and therefore, their datacenter will be virtualized and the IT operating fabric will be grid-based, tapping the power of external grids of CPU.
7. RFID Everywhere. The Z Generation will be the ones that take RFID to new heights. Everything that is taggable will be tagged and tracked.
8. CIO and Z Generation IT Professionals will leverage the power of Internet video by taking advantage of companies like BrightCove Networks which will bring knowledge workers engaging channels like "The Customer Service Channel", "The Corporate Strategy Channel", "The M&A Channel" and others. I can see companies like Harvard Business Review and others producing content for these channels. Imagine the "The DataCenter Channel." The topics are endless and will be as easy to find as bringing up your favorite search engine. New content will be generated targeted for companies like "The IBM Channel" or the "GE Channel." I would love to see "The Institute of The Future Channel"
9. Intelligent Wikis will be the primary source of knowledge in an enterprise and will eventually do what data warehousing did for business intelligence. Furthermore, new internal employee-generated communities will spin up to voluntarily invent new projects during their off-hours to showcase their creativity that is typically not known by the employer.
10. CIOs will aggressively adopt Prediction Networks as part of the core business strategy to better help the enterprise gauge where everything from sales to new product development will be successful.

2. ***The convergence of BPM and Web 2.0 begins.*** BPM is about improving performance by optimizing key processes. Web 2.0 is about capturing the wisdom of crowds (or as O'Reilly puts it, the architecture of participation). The convergence of BPM and Web 2.0 enables collaborative development and tagging of sub-processes, establishing a "process folksonomy" where the best processes can evolve organically. Collaboration can occur over simple but highly scalable pub/sub mechanisms (like Atom or SSE). Lightweight tools will enable users to model or reuse sub-processes using a broad set of metadata. While this is an exciting opportunity, there are several technical and non-technical issues that must be addressed before this convergence becomes a reality.

3. ***Improvements in SOA management and governance.*** Tools, frameworks and platforms will emerge that better enable:
   - Defining and enforcing service development guidelines
   - Modeling, managing and enforcing operational policies (e.g., security, service level agreements and others)
   - Service simulations (what-if scenarios, impact analysis, etc.)
   - Modeling and managing service dependencies
   - Service provisioning and de-provisioning
   - Configuration management

4. ***Workflow isn't confined to the datacenter anymore.*** Lightweight, extensible frameworks like Windows Workflow Foundation (WF) enable workflow in places where it may not have been previously considered.

5. ***Better UI Experiences.*** Declarative user interfaces will enable rich user experiences that can be easily modified or extended with simple mechanisms like XSLT. Familiar business applications like Office provide the user interface to back-end line-of-business systems. The line between AJAX-based UIs and rich desktop UIs will blur, enabling users to enjoy both connected and occasionally-connected experiences. Tools and guidance will make building, testing and deploying these composite UI experiences much easier.

6. ***A new category of architecture emerges: Software + Services.*** It is hard these days to find an architectural concept that is *not* some how tied to services. The line between Web services, SaaS and traditional applications will blur to the point where the location, contract and hosting of a service are less important than the capabilities exposed by the service.

7. ***JSON without AJAX.*** We'll start to see more people using JSON to address the XML bloat problem outside of simple AJAX-based applications. The downside is that this may result in more tightly-coupled applications.

8. ***Events and states instead of EDI-style messaging.*** Lightweight frameworks will empower developers to starting thinking about solutions in terms of event notifications instead of simple messages passing from point A to point B. Hierarchical state machines enable state synchronization across complex, federated processes.

9. ***We stop talking about SOA and "just do it."*** Sometimes we spend more time arguing about IT trends than actually using them. In 2007 the tools and specifications we need for enterprise-strength, loosely-coupled solutions have finally arrived – it's time to roll up our sleeves and get to work.

10. **IT finally admits that there is no silver bullet.** Every year I hope to see this happen and every year my hopes are crushed by buzzword-of-the-minute hype machines. *(Hey I can dream, can't I?)*

AJAX Over-use • JSF • Relational Object Mapping • Macs

BILL DUDNEY
Editor-in-Chief, *Eclipse Developer's Journal*

1. AJAX will continue to gain momentum as folks continue to have the epiphany that Web 1.0 UI is not good for users. Overuse of the technology will be a real problem. JSF will finally start to become a de facto as well as actual standard due to its ease of integration with AJAX.

2. Open Source enablement will continue to be a hot spot for VC investment. I don't think the perfect business model will emerge in '07 though so the market will still remain 'immature.'

3. Java Persistence API will bring relational object mapping to the long tail of the market. Early adopters will be wondering what all the hype is because the technology is so old in their eyes.

4. Macs will continue their 'thought leader' adoption curve. This is not the year they start to penetrate the corporate IT department.

Web Service Orchestration • Web Services Explosion

ADAM KOLAWA
Co-Founder & CEO, *Parasoft*

1. I anticipate a significant demand for Web service orchestration in the upcoming year, especially in the United States.

   Many organizations now have at least one Web service, and a growing number already have two or more related Web services. Managing multiple related Web services is considerably more challenging than manag-

ing the same number of separate, unrelated Web services. To use these related Web services to achieve your business goals, you need to consider how high-level operations pass through the Web services, then determine how to implement this high-level flow— from start to finish. This can be accomplished in two ways:

- By programmatically coding the application logic required to tie the involved elements together.
- By using an orchestration tool to direct the flow through the involved elements, which remain separate.

I predict that the latter method will be the favorite because it is easier.

2. I also expect an explosion of Web services because they are so easy to expose. Once exposed, Web services basically create interfaces which can be reused. This will significantly reduce the amount of code that needs to written, which will in turn cut the demand for "bare bones" development.

Server Virtualization • Container-Based Hosting • Linux Rails • Django • Agile Development

BRANDON HARPER
Senior Software Developer at *Acxiom Corp.*

The top five technology trends I see happening in the New Year are:

1. Server virtualization is just getting started, and will really make itself known in the coming year. Once we start seeing the quad core CPU architectures as a part of standard infrastructure, it really starts making a lot of sense to start deploying and managing servers and applications as virtual entities rather than specific pieces of hardware. This helps manage the cost and pain of software configuration management, take advantage of being able to process many tasks simultaneously because of hardware support, as well as allows legacy hardware to be retired in favor of applications running on virtual servers.

2. Container-based hosting is the new kid on the block, and will also start making its presence known in the upcoming year. Commonly labeled as "grid" hosting (which is a technical misnomer if you understand distributed computing), it essentially claims to be an infinitely scalable hosting platform. This technology still seems to be half-baked at the moment, but you could have said the same thing about Linux ten years ago.

3. People who normally wouldn't use Linux start to explore it and even replace Windows with it permanently. With Vista, Microsoft seems to be moving to a model in which the Windows operating system is a method to police users with DRM and other nonsense rather than provide developers with a good platform on which to use hardware, which is what operating systems are really supposed to be. A lot more consumers who haven't noticed this happening in the past will stand up and notice this year.

4. Dynamic languages and frameworks will continue to make leaps in popularity and adoption. Given the current squeeze on technology talent in the US, companies are going to have to learn how to do more with fewer resources. Moving to dynamic languages and frameworks as well as other simplification such as varying Agile software development practices will enable this to take place. I think the obvious leading candidates here are Ruby on Rails and Django.

5. The enterprise will embrace ways to simplify development by continuing to embrace open source software and Agile Development strategies. While there are a lot of cries to the effect of Ruby on Rails replacing Java, I think that's complete nonsense as Java is a language and Ruby on Rails is a framework. Rapid development languages will certainly make some inroads, particularly where heavy tools have been used to build simple applications, Java is still going to be a major part of the service-oriented enterprise for years to come because of the power and tools it provides as well as its industry support.

Open Sun • iPod Uno • IT2 • Microsoft VAPOR

...And in Other 2007 News
by Richard Monson-Haefel
Award-Winning Author & Senior Analyst, Burton Group

1. ***Jonathan Schwartz open-sources Sun Microsystems.***
In a move that will surprise everyone Sun Microsystems will announce that it will open source its entire company. Sales, marketing, finance, and even operations will be open to the community for anyone to contribute.

2. ***Apple computer announces the iPod Uno.***
The size of a match stick with no screen or controls, the iPod Uno plays one song in a constant loop. Despite its limited capabilities, the tiny device becomes an instant hit and a cultural icon.

3. ***In what is heralded as the seminal article on the subject, Tim Berners-Lee mentions "IT2"***
Overnight the term morphs into "IT 2.0," spawning thousands of blog entries and press articles, a dozen books, five conferences, and millions of dollars in venture capital. It turns out that the original article, incomprehensible to most readers, was actually another attempt to explain the Semantic Web and the IT2 reference was just a typo.

4. ***Microsoft will create the first CMO (Chief Marketing Officer) position.***
The new CMO will immediately change his own title to Chief Command & Control of Packaging Officer (C3PO) and then announce that Vista will be delayed and renamed Microsoft Virtualization Application Program Operating system Reloaded (Microsoft VAPOR).

Oracle JDeveloper – An I²DE Worth a Second Look

It's never too late for a second chance at a first impression

Reviewed by
Lucas Jellema

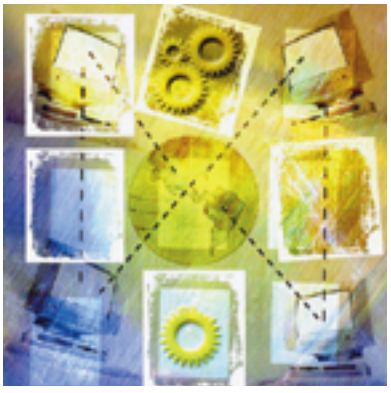
As the saying goes you never get a second chance at a first impression. In general, that's true, but if you've been thoroughly revitalized, matured, and cosmetically re-engineered, shouldn't you get a second shot at that first impression? I'd argue that's true of Oracle's Java/J2EE Workbench, Oracle JDeveloper.

Starting out life as a code fork from Borland's JBuilder tool back in 1997, Oracle JDeveloper has had a fairly long history as a Java IDE, most of it in relative obscurity. Things have changed though. During JavaOne 2005, Oracle announced that Oracle JDeveloper would henceforth be free. This sparked interest, since JDeveloper always had areas of definite benefits over other tools but came with a price tag. Furthermore, over the last couple of years, Oracle has added rich support for all areas of Java and JEE development, especially in the case of the latest 10.1.3.1 rev released in October 2006 during Oracle OpenWorld.

This latest release supports JEE 5 standards such as EJB 3.0 and JSR-220 Java Persistence API, JSR-181 Web Service Annotations, as well as JavaServer Faces. It also brings a visual design time environment for creating Business Process Execution Language (BPEL) processes and Enterprise Service Bus (ESB) Services and features-enriched functionality for Java, XML, Web Services and Web application development. I'd like to invite you to take a brief tour to get that second look and a new first impression of Oracle JDeveloper.

Getting To Know Oracle JDeveloper

JDeveloper 10.1.3.1 can be downloaded from Oracle's Technology Network at http://oracle.com/technology//jdev (up to 420 MB). Installation only takes unzipping an archive. When running, JDeveloper's Studio Edition consumes a somewhat greedy 180 MB memory by just being there.



Plain Old Java Programming

Java IDEs are typically evaluated first and foremost on their ability to support pure Java programming. Oracle JDeveloper has a history riddled with wizards and frameworks and facilities for all kinds of development, but it tended to trail behind other prominent IDEs when it came to Plain Old Java Programming (POJP). With recent releases that gap has been closed and Oracle JDeveloper currently offers at least the same, if not more features, than can be found in alternative tools.

Built-in features include:

- Refactoring – with options such as extract code fragment as method and extract interface or superclass
- Code folding
- Quick view of JavaDoc of referenced classes and methods
- Code completion and code reformatting
- User-customizable code snippets
- Generation of bean accessors for properties
- Smart import organization of classes and libraries

One feature I particularly like and frequently use is the CTRL+ – a key combination for "Go to Java Class." This brings up a window where developers can type in the name of a class or interface and navigate directly either to the source code or the JavaDoc.

Advanced tools found in Oracle JDeveloper, not generally part of the core of IDEs, include a code profiler to identify performance and memory hotspots, and an auditor that does quality control on Java source code – even on non-compilable code – according to predefined standards and guidelines. The profiler monitors and logs a running program's use of processor and memory resources and can be used to locate and correct inefficiencies. Developers can also use the profiler with the debugger and CodeCoach for efficient source code troubleshooting.

The march of Plain Old Java Objects (POJOs) seems unstoppable. With EJB 3.0, and particularly the Java Persistence API (JSR-220), as well as the Web Service annotations defined in JSR-181, a POJO can be easily promoted to an entity, persistently mapped to a database table or a Web Service simply by adding some annotations to the bean definition. JDeveloper will recognize these annotations and provide code completion support for them.

JDeveloper 10.1.3.1 also features wizards that can create entities (annotated POJOs) from selected tables and views in a database, or that can create a new entity. Currently there's no synchronizing support to realign entities and tables; however, given Oracle's leading role in the Dali Eclipse plug-in, I expect this feature to be added to a future version of Oracle JDeveloper.

Lucas Jellema (Oracle ACE) is CTO at AMIS, an Oracle, Java and SOA Technology Consulting firm based in Nieuwegein, The Netherlands. Apart from being a technical architect and workshop instructor, he is a regular presenter at international conferences on topics such as BPEL, EJB 3.0, AJAX and Java Server Faces, Oracle's ADF (Application Development Framework) and productive application development in general.

Lucas.Jellema@AMIS.nl

The Web Service wizard allows easy publication of a POJO and selected methods as a Web Service; this wizard can add the JSR-181 annotations to the POJO, or create a WSDL document along with the WSIF binding definition.

Integration IDE

The biggest news with Oracle JDeveloper 10.1.3.1 is the Integrated Service Environment workbench. In particular, BPEL and ESB tooling is now fully integrated into JDeveloper.

There is a visual diagrammer with drag-and-drop component palettes and built-in wizards that help the developer design the BPEL processes. Wizards help create partner links for adapter services that link to external systems like (SOAP) Web Services, JMS, file system and FTP servers, MQ Series and databases (SQL or Oracle PL/SQL). The BPEL process is created by dragging BPEL activities such as pick, flow, invoke, reply, and assign to the diagram and configuring them through wizard screens. The invoke and reply steps are connected to the partner links with external services. A BPEL process can be deployed directly to the Oracle BPEL Process Manager from within the JDeveloper IDE (see Figure 1).

Oracle JDeveloper can also generate test cases for BPEL processes in which partner link response messages and workflow outcomes can be emulated prior to deployment in a production environment. This helps ensure that a process interacts with Web Service partners as expected by the time it's ready for deployment to a production environment.

In late October 2006, Oracle released its ESB as part of its SOA Suite. Oracle JDeveloper provides the design time for the ESB, and ESB router services typically consist of inbound services, routing and transformation rules and outbound services. The router service is constructed visually, using drag and drop, as well the same adapter service wizards used for developing BPEL processes. The transformation of messages in the ESB is done using an XSLT transformation. Oracle JDeveloper has a particularly useful tool that makes creating the XSLT document a simple, highly visual task, using drag and drop from source (inbound) XSD to the target XSD document.

Deploying the ESB is a two-click process using a predefined connection to the application server that hosts the ESB.

The first impression in developing ESB services is that it works very well – even though this is just a 1.0 release of the technology.

I²DE – Integrated IDE

Some integrated development environments (IDE) are more integrated than others. Oracle JDeveloper is much more than a Java programming tool. Integrated into its core Java IDE are a host of other IDEs such as:

- ***XML development*** – visual editors for XSD, XSLT, XQuery, WSDL, and support for debugging XSLT
- ***Web development*** – visual editors for CSS, as well as WYSIWYG editors for HTML, JSP, JSF (pages and config) and ADF Faces (a k a Apache MyFaces Trinidad), Struts (config) and Applets and an HTTP Analyzer for analysis of the packets sent across the wire for Web Services and Web applications
- ***J2EE development*** – wizards for EJB, Web Services, and the built-in OC4J Application Server to deploy J2EE artifacts quickly, as well as very easy remote debugging of both Web and J2EE applications. JDeveloper also provides an IDE for Oracle TopLink – one of the premium tools for object-relational mapping
- ***Database development*** – editors and diagrammers for tables, database browsers, SQL worksheets, data viewer for all JDBC-powered databases as well as programming and debugging support for PL/SQL – the Oracle database's stored procedural language
- ***UML modeling*** – diagrammers for activity, class (with code synchronization), sequence and use case diagrams
- ***Integration*** – visual modeling, testing. and deploying Web Services, BPEL processes, and ESB services

These various IDEs work together and are blended in the overall IDE. Together they share connections (database, application server, UDDI, WebDAV), a project definition with path-setup, library associations, and deployment profiles. Generic tools include a property palette, a structure window, debugger, and a task manager. JDeveloper integrates with various application servers – WebLogic, Tomcat, JBoss, OC4J, and Oracle Application

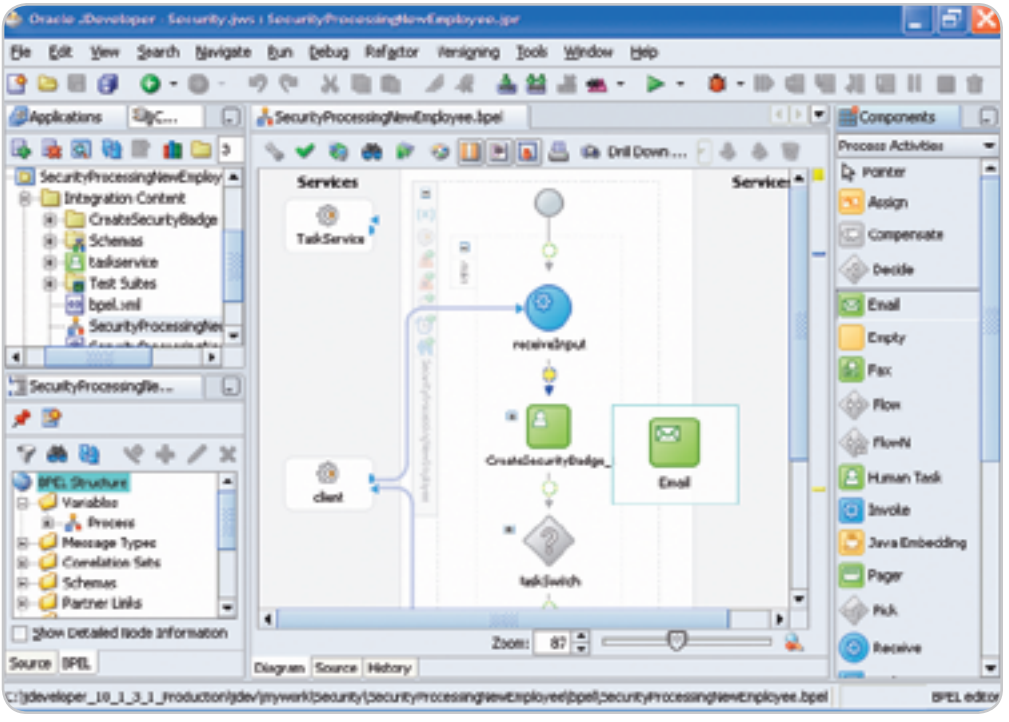


Figure 1 BPEL Designer

> Oracle JDeveloper has functionality that extends far beyond what one would normally expect from a Java IDE and needs to escape the prison of its past"

Server for one-click deployment, and also provides powerful Ant integration (see Figure 2).

Check for Updates/Extensions

No matter how rich an IDE may be it's never complete out-of-the-box. Like other IDEs, Oracle JDeveloper has an automated mechanism for installing extensions – the JDeveloper term for what other IDEs call plug-ins – that's also used in updating the IDE itself with patches and service updates.

However, there's obviously only a limited set of extensions available for Oracle JDeveloper, since most of the functionality is *already* built-in and pre-integrated. While that may mean a little less choice, it most certainly saves a lot of time and money otherwise spent on acquiring the collection of plug-ins offering the same functionality only to find them far less well integrated than one would hope for.

Some useful extensions – that you might have expected to come pre-integrated – include unit testing with JUnit and support for AspectJ and Subversion.

Installing and upgrading extensions is effortless; however, downgrading or de-installing extensions requires developers to disable the extension and remove the archive from the file system.

Oracle Frameworks for Productive Java Development

Oracle JDeveloper is frequently known for its built-in frameworks, such as ADF Business Components – a SQL-oriented framework for mapping between Java applications and relational databases – and ADF Model – a data-binding infrastructure based on JSR-227. While these frameworks can add tremendous productivity, such as the drag-and-drop development of a database-bound JSF application, they are often regarded with some suspicion, because people see them as too proprietary and closed.

Every organization needs to make its own judgment about using these frameworks. They should take into account that these frameworks lower the barrier considerably for doing (productive) J2EE development, allowing less-experienced developers to make substantial contributions to development efforts. Furthermore, Oracle is using these frameworks to develop Fusion Applications, virtually guaranteeing their continued support and enhancement. Applications developed with these frameworks are J2EE-compliant and can be installed on various application servers.

Conclusion

The new 10.1.3.1 release of Oracle JDeveloper has a lot to offer with functionality that extends far beyond what one would normally expect from a Java IDE. One of its key strengths is its out-of-the-box richness, comprehensive feature set, and tight integration. No need for searching, acquiring, and installing a lot of plug-ins.

In the past there's been a lot of prejudice concerning Oracle JDeveloper: not being suitable for Plain Old Java development, only supporting proprietary Oracle development, being too expensive, etc. However, Oracle JDeveloper 10.1.3.1 is free, allows – but by no means necessitates – using Oracle-specific frameworks, and has scores and scores of features that even the most hardcore Java programmer will appreciate. Support for SOA, XML, Web, database, UML, and J2EE is an added bonus.

If you haven't looked at Oracle JDeveloper in a while, you'll definitely want to take a second look now. Installation only takes unzipping an archive. Oracle JDeveloper 10.1.3.1 is free and available for download at: http://oracle.com/technology//jdev.

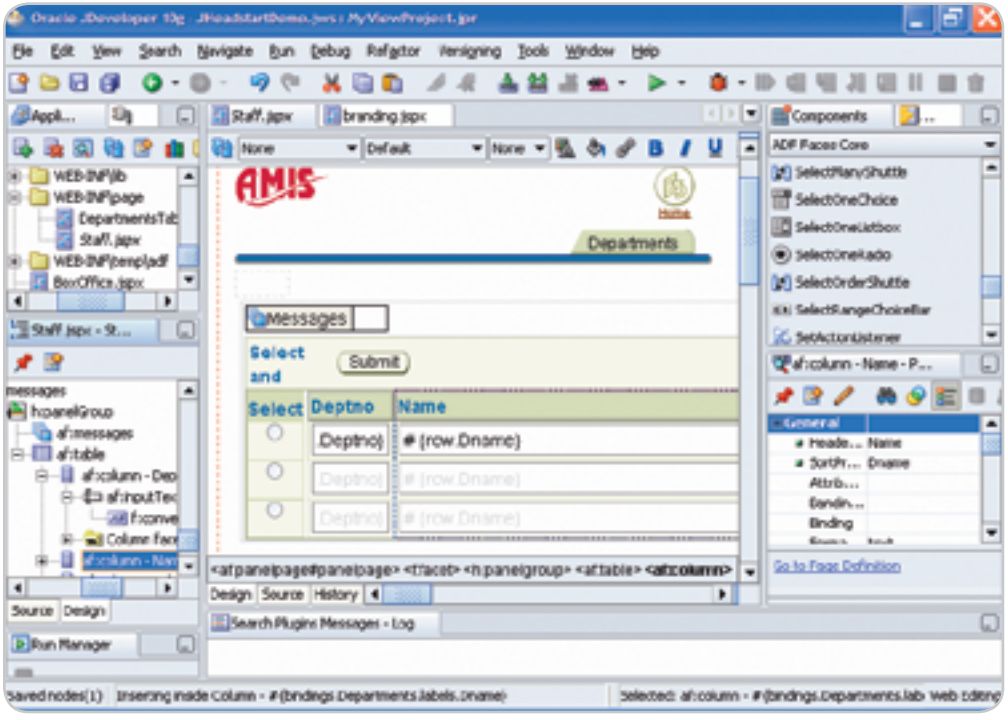


Figure 2 ADF Faces Visual Editor + Structure Window, Property Palette, Component Palette

JAVA DEVELOPER'S JOURNAL Advertiser Index

Advertiser	URL	Phone	Page
AjaxWorld East Conference 2007	www.ajaxworldexpo.com	201-802-3022	51
AjaxWorld University Bootcamp	www.ajaxworldbootcamp.sys-con.com	201-802-3022	49
Altova	www.altova.com	978-816-1600	4
Backbase	www.backbase.com/jsf	866-800-8996	21
Business Objects	www.businessobjects.com/devxi/misunderstood		11
IBM	ibm.com/takebackcontrol/flexible		7
ICEsoft Technologies	www.icesoft.com	877-263-3822	39
Infragistics	www.infragistics.com/jsf	800-231-8588	8-9
Instantiations	www.instantiations.com/rcpdeveloper/resources/casestudy-bea.pdf		32-33
Java Developer's Journal	www.jdj.sys-con.com	888-303-5282	61
Jinfonet Software	www.jinfonet.com/live	240-477-1000	27
Northwoods Software Corp.	www.nwoods.com	800-434-9820	41
OPNET Technologies, Inc.	www.opnet.com/panorama	240-497-3000	13
Parasoft Corporation	www.parasoft.com/jdjmagazine	888-305-0041	Cover IV
Quest Software	www.quest.com/hero	949-754-8000	Cover II
Recursion Software	www.recursionsw.com	800-727-8674	29
Software FX	www.softwarefx.com	800-392-4278	Cover III
TIBCO Software Inc.	http://developer.tibco.com/	800-420-8450	17

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

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

The 2006 JCP EC Elections Are Over

Meet the newly elected and re-elected members

Onno Kluyt

Congratulations go this year to IBM; Oracle, HP; Fujitsu; Doug Lea, professor of computer science; Motorola; Vodafone; Siemens; BenQ; Ericsson AB; and Jean-Marie Dautelle, individual developer and initiator of several open source projects. The first four are now re-elected on the SE/EE EC for another three-year term as a result of the Ratification Ballot and the fifth as a result of the Open Nominations/Election Ballot.

Representing IBM on the EC, Mark Thomas leads the development teams in providing IBM software developer kits for Java technology. His development experience includes graphics, windowing systems, message queuing, computer-aided telephony, voice response systems, and Java technologies. Don Deutsch, who stands in for Oracle on the EC and other standards boards, is the 2002 recipient of the Edward Lohse Information Technology Medal for his leadership of national and international information technology standardization. Scott Jameson, representing Hewlett-Packard on the EC and other standards organizations, is currently chairman of ISO/IEC JTC 1, Information Technology. Masahiko Narita of Fujitsu Limited serves as primary representative on the EC. He has actively promoted object technology in the Japanese market.

Doug Lea is professor at the computer science department at the State University of New York at Oswego. He authored and co-authored many books and articles about Java and object-oriented systems. Within the JCP, Doug was the Spec Lead for JSR 166, Concurrency Utilities, and has served as an Expert Group member on most JSRs dealing with core Java SE for the past five years.

The runners up in the SE/EE open elections stage were:

- ***Capgemini***, a full-service IT provider, employs thousands of Java developers. The company joined the JCP community in 2004, served actively on the Java EE 5 and Java SE 6 Expert Groups, and supported the Java Portlet specification and the release of JBI.
- ***Tom Crosman*** follows Java technology very closely – its history, current status, and future, having worked full-time on Java since the launch of JDK 1.0.1. He has commented on JSRs and voted in JCP elections.
- ***Jean-Marie Dautelle*** is an individual developer with affiliations and contributions to open source projects. More about Jean-Marie as winner of the ME EC race.
- ***Justen M. Stepka*** worked with a team to develop, market, and support the IDX sign-on and identity management framework when he was CEO of Authentisoft, recently purchased by Atlassian. He is a contributing author for O'Reilly and other popular publications and a speaker at JUGs and TheServerSide.com conferences.
- ***Evan Summers*** is a Java developer working on foundation classes to reuse in future projects and applications, possibly involving Swing clients using RESTful Web services. He currently participates in JSR 295, Beans Binding, and JSR 296, Swing Application Framework.
- ***Mauro Do Valle*** was a JUG leader in Rio Grande do Sul, Brazil, for the past four years, doing many events and projects for the Java Community. He joined the JCP program as an individual three years ago.

I'd like to thank all runners up for participating in the 2006 elections and for their strong interest in the JCP. The other members of the SE/EE are Apache Software Foundation, BEA Systems, Borland, Google, Intel, Nortel Networks, Red Hat Middleware LLC, SAP, SAS Institute Inc., Hani Suleiman, and Sun Microsystems. For information on these representatives, go to http://jcp.org/en/participation/committee.

The other elects – Motorola, Vodafone, Siemens, BenQ, Ericsson AB – are now representatives on the ME EC for three-year terms, or in the case of the second highest voted candidate on the ME EC, Jean-Marie Dautelle, for two years (more regarding EC terms in Q/A14 at http://www.jcpelection2006.org/faq/).

Motorola's primary representative, James Warden, has over 25 years of experience developing software and system architecture for mobile and embedded devices. He is the Maintenance Lead for JSR 118, Mobile Information Device Profile 2.0, and has served on 10 other Expert Groups. Guenter Klas, representing Vodafone, leads the company's terminal standardization program, coordinates the efforts of Vodafone's Spec Leads and Experts, and through his team is involved in standards organizations such as the W3C, the Open Mobile Alliance, Global System for Mobile Communications Association, and Open Mobile Terminal Platform. Siemens AG is represented on the ME EC by Lothar Borrmann, who leads the Software Architecture department in assisting and advising Siemens' Groups regarding software architecture issues, innovations, and trends, including software platform technologies such as the Java technology. At the time of writing, BenQ Corporation is still in the process of appointing a primary representative.

Ericsson AB and Jean-Marie Dautelle came out the winners of the open nomination. Magnus Olsson of Ericsson Mobile Platforms represented Ericsson AB on the ME EC before. He worked with mobile telephony (GSM) infrastructure development, mobile terminals, and Ericsson Mobile Platforms (EMP) incorporating Java ME technology into the applications domain – system design, protocols, testing, and application development. Jean-Marie Dautelle is Java SE 5.0 Certified, the initiator and primary developer of two popular open source projects: Javolution (http://javolution.org) and JScience (http://jscience.org). JScience provides the current Reference Implementation for the Expert Group Jean-Marie serves on – JSR 275, Units Specification.

And the runner up in the ME EC open nominations race was SiRF. By many accounts SiRF is among the fastest-growing companies in Silicon Valley and among market leaders in location technologies and video for mobile handsets. Within the JCP community, SiRF has actively participated in JSR 293, Location API 2.0; JSR 281, IMS Services API; and JSR 298, Telematics API for Java ME.

I'd like to thank SiRF for participating in the open nominations for the JCP ME EC. The other members on the ME EC are IBM, Intel, Nokia, NTT DoCoMo, Orange France, Philips, Research In Motion, Samsung, Sony Ericsson, and Sun Microsystems.

More details about the ECs' members and their Java technology and community expertise are posted on jcp.org at http://jcp.org/en/press/news/ec-feature_SE091206 for SE/EE EC and at http://jcp.org/en/press/news/ec-feature_ME091206 for ME EC.

For a stage-by-stage navigation of the 2006 JCP EC elections results, go to the official JCP Elections page hosted by PricewaterhouseCoopers at http://www.jcpelection2006.org/jcp/overview.

Join me in congratulating the newly elected and returning EC members and in wishing them a successful term ahead.

Onno Kluyt is the director of the JCP Program at Sun Microsystems and Chair of the JCP.

onno@jcp.org

Complex and evolving systems are hard to test...

Parasoft helps you code smarter and test faster.

Start improving quality and accelerating delivery with these products:

Awarded **"Best SOA Testing Tool"** by Sys-Con Media Readers	**InfoWorld's 2006 Technology of the Year pick** for automated Java unit testing.	Automated unit testing and code analysis for C/C++ quality.	Memory errors? Corruptions? Leaks? Buffer overflows? Turn to…	Easier Microsoft .NET testing by auto-generating test cases, harnesses & stubs	Automate Web testing and analysis.
SOAtest™	*Jtest*™	*C++test*™	*Insure++*™	*.TEST*™	*WebKing*™

PARASOFT®

We make software work.™

Go to www.parasoft.com/JDJmagazine • Or call (888) 305-0041, x3501