SYS-CON

AllFusion® ERwin® Data Modeler – "Readers Choice Winner" Voted "Best Database Tool or Driver" by readers of Java Developer's Journal, in your hands FREE – try it today!

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

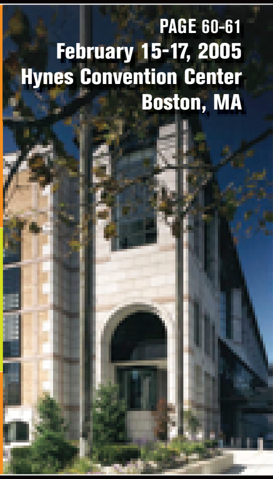
# JDJ

NOVEMBER 2004 VOLUME:9 ISSUE:11

web services EDGE conference & expo

- Join Over 3,000 Developers
- App Server Shoot-Out!
- FREE J2EE Tutorial
- JDJ Partner Pavilion
- Seminars and Case Studies

$1,000,000 Software Giveaway



## BUILDING MANAGEABILITY

*Using the management and monitoring APIs to build application manageability*

PLUS...

- Creating a Pet Store Application
- Understanding Portals and Portlets
- EJB 3.0 Preview
- Java on Wall Street
- J2SE 5.0 Ready for Business
- Distributed Notification with Java RMI

RETAILERS PLEASE DISPLAY UNTIL JANUARY 31, 2005

$5.99US $6.99CAN

12

0 09281 01751 6

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

## Oracle JDeveloper

# "Second to none."

*— July 5, 2004, eWEEK*

eWEEK LABS ANALYST'S CHOICE

**July 5, 2004**
**Oracle JDeveloper 10*g***

ORACLE®

**oracle.com/tools**
**or call 1.800.633.0759**

**Subscriptions**

For subscriptions and requests for bulk orders, please send your letters to Subscription Department:

**888 303-5282**
**201 802-3012**
**subscribe@sys-con.com**

Cover Price: $5.99/issue. Domestic: $69.99/yr. (12 Issues) Canada/Mexico: $99.99/yr. Overseas: $99.99/yr. (U.S. Banks or Money Orders) Back Issues: $10/ea. International $15/ea.

**Editorial Offices**

SYS-CON Media, 135 Chestnut Ridge Rd., Montvale, NJ 07645
Telephone: 201 802-3000 Fax: 201 782-9638

Java Developer's Journal (ISSN#1087-6944) is published monthly (12 times a year) for $69.99 by SYS-CON Publications, Inc., 135 Chestnut Ridge Road, Montvale, NJ 07645. Periodicals postage rates are paid at Montvale, NJ 07645 and additional mailing offices. Postmaster: Send address changes to: Java Developer's Journal, SYS-CON Publications, Inc., 135 Chestnut Ridge Road, Montvale, NJ 07645.

# Sung and Unsung *i*-Technology Heroes

Jeremy Geelan

When I asked in a previous editorial who the Top Twenty Software People in the World were, I knew there would be a widely divergent response from readers. As promised, here's a preliminary update on the identity of some of your nominees.

The exercise was promoted, if you recall, by XML pioneer Tim Bray, who nominated as a "Top Twenty" candidate Adam Bosworth, famous for Quattro Pro, Microsoft Access, and Internet Explorer 4 even before he went on to become BEA's chief architect before recently leaving the Java app server company to join Google, Inc.

Readers very soon nominated another Google staffer (since 2002), namely Rob Pike, an early developer of Unix and the windowing system (GUI) technology, and a long-serving member of the Unix team at Bell Labs before he joined Google.

Pike joined Bell Labs in 1980, the same year he won the Olympic silver medal in archery. Since he received several nominations, there is clearly something deeply appealing about a world-class archer going on the following year to hit the bull's eye by writing the first bitmap window system for Unix systems (since then he has written 10 more).

Pike became well known for his appearances on "Late Night with David Letterman," which ranks him, profile-wise, right up there with Linus Torvalds, another nominee (no surprise there).

Hero of the open source movement, geek made good, thorn in Bill Gates' side, Torvalds, it has been said, "embodies the idea that there is always another way, an antidote to the Microsofts of this world, evidence that the idea of the 'community' within IT is still there." As one wit expressed it, "If it wasn't for the presence of Lara Croft and Xena Warrior Princess, techies around the world would have posters of Linus on their walls."

Anders Hejlsberg, another nominee originally from Scandinavia, is the Danish-born genius associated with Turbo Pascal, Delphi, C#, and the Microsoft .NET Framework. Hejlsberg is one of the industry's most charming and modest high achievers – a "serial success" and a worthy nominee.

Arthur van Hoff, the programming legend who now works at TiVo, was another early nominee. One of the Java geniuses at Sun (he is said to have almost as many patents as Bill Joy), in 2002 van Hoff started Strangeberry, which TiVo bought in January of this year with the aim of ensuring that next-generation TiVos will be able to recognize Web content and direct it to the appropriate home device.

Sam Ruby, nominated for being "probably the most widely read blogger on the planet," is a 21-year veteran of IBM who has been hailed as a "Web services visionary." Ruby is a member of the board of directors and vice president of the Apache Software Foundation; a developer on the Apache SOAP project; chairman of the Jakarta project; and a member of the PHP group, a select group of developers who contribute to core PHP.

Some of the most interesting nominations thus far have been of the "unsung heroes" of *i*-technology – or the less-sung ones, if you will. Take for example one of the three creators of XP, Kent Beck. Author of the first book on the topic, *Extreme Programming Explained*, published in 1999, Beck was nominated by a reader: "For (arguably) pioneering, and certainly ruthlessly promoting, the notion that change (specifically, to requirements of commercial software) must be embraced (actively designed for), not avoided, and that trying to design everything once and for all up-front is an exercise in futility."

There are those who believe excellence is not a skill but an attitude. Still others say that excellence is in the details. Whatever the truth, and it's most likely a mix of both, the search will go on. So far no fewer than 40 individuals have been identified by readers as possible candidates for the top 20 positions. Feel free to keep nominating; by the end of the year we shall be able to draw up the definitive list of whom readers consider to be The Top Twenty Software People in the World. The address for nominations, again, is toptwenty@sys-con.com.

**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*

The new stud on the server farm.

Presenting the new Xserve® G5, a wickedly fast, extremely compatible and refreshingly affordable 1U server from Apple®.

With dual 2GHz 64-bit G5 processors, it achieves blazing speeds of up to 30 gigaflops. It's so powerful, in fact, that the U.S. government is deploying* 1,566 Xserve G5 servers to create one of the world's fastest supercomputers, capable of up to 25 trillion calculations per second.

And it comes complete with Mac OS® X Server, Apple's UNIX-based operating system that provides a complete suite of standards-based network services with no per-client fees. So whether you have Mac®, Windows, UNIX or Linux clients, Xserve is ideal for cross-platform file sharing, hosting dynamic websites, streaming audio and video and running powerful J2EE applications – right out of the box.

Of course, its most impressive feature may be its price, starting at just $2,999†. The new Xserve G5.

# Solving the Toughest XML Data Integration Challenges!

Free 30-day Trial Download Now!

**Simplify your XML Development with Stylus Studio XML IDE**

STYLUS STUDIO

## Introducing Stylus Studio 6 XML Professional Edition

Tired of the complexities of advanced XML data integration? New and improved **Stylus Studio 6** washes away even the most stubborn data integration problems with a bold new formula that includes such active ingredients as **Convert to XML™**, a powerful utility for creating XML documents from any flat file data source. Support for **XSLT 2.0** and **XQuery 1.0** in Stylus Studio's leading XML editors, debuggers, and performance profilers. An all-new **XML Schema Editor** for visually developing XML data models. An **XML Grid View** that lets you work with tabular data as XML. Usability enhancements to our **XML Mapper**, and much more.

**Stylus Studio XML IDE.
Revolutionizing the way XML data integration gets done.**

XML Productivity Through Innovation. www.stylusstudio.com

NOVEMBER 2004 VOLUME:9 ISSUE:11

# JDJ contents

JDJ (ISSN#1087-6944) is published monthly (12 times a year) for $69.99 by SYS-CON Publications, Inc., 135 Chestnut Ridge Road, Montvale, NJ 07645. Periodicals postage rates are paid at Montvale, NJ 07645 and additional mailing offices. Postmaster: Send address changes to: JDJ, SYS-CON Publications, Inc., 135 Chestnut Ridge Road, Montvale, NJ 07645.

# Have You Checked Your Basement Lately?

Venkat

Many of you have been living in the same house for several years. In the process you may have accumulated furniture, clothes, antiques, etc., and have forgotten why certain things were purchased in the first place or that certain things even exist in your basement.

Large enterprises have large basements. Their limited space is overflowing with existing hardware, software, and applications. Why is it that as technology has advanced and newer systems are being purchased, many companies still have the 30-year-old mainframe and its related technologies in their IT basement? I'm not suggesting that you throw out the mainframes. The more technology that comes out each year, the more everyone is buying and piling it in their basement without checking to see if they need it and whether an existing system needs to be eliminated first. And how will the newly purchased products work in conjunction with existing systems? Java technology, introduced into an enterprise without a plan, compounds the existing chaos as anything else. The Java platform includes some APIs to help enterprises alleviate the problems and manage their applications but these APIs are rarely utilized.

If you are organizing your basement, creating space and maybe thinking of buying new stuff to fill the newly created space, this time ensure that there is some way to keep inventory. Track applications, services, and components through a knowledge repository. Ensure that what you buy works well with what you already own. Last of all, manage everything you own so you can finally track and realize the much sought after ROI.

Renovation, integration, and management – these are the three stages needed to organize an enterprise's IT landscape.

## Renovate

The renovation stage is the least automated of the three. When renovating, an enterprise needs to take a hard look at what they own and decide if there are similar functionalities in multiple places, whether certain tools are redundant, and whether certain applications are even being used. It involves manually sifting through the IT landscape and compiling a knowledge base. I've come across situations where applications that were not being used were still being supported in a production environment.

A catalog of applications, components, and services needs to be created with an enterprise-wide metadata repository containing attributes about these services. The repository also needs to include policies, standards, and guidelines for building or buying new stuff. A simple repository and knowledge management tool can be created using Java technology like JSP and servlets, the Java Metadata API, and the Web services–related APIs. This can be searchable and modifiable and gives a unified view of the IT landscape.

## Integrate

The integration stage involves making architectural decisions and understanding relevant scenarios. The Java technology platform has strong support for messaging and Web services including service description and discovery, security, and distributed computing. There are also various pure play, multi-protocol enterprise service bus platforms based on Java that have emerged that allow integration in a heterogeneous environment. The ESB platforms are branded as business integration platforms. They support various messaging architectures and include support for JMS, JCA, and SOAP. The Java Connector Architecture is an integration mechanism that is part of the J2EE specifications to integrate to enterprise information systems.

## Manage

An enterprise needs to factor in management of their components and services early on in the architecture and design phase. The Java platform provides Java Management Extensions (JMX) as a part of its core. I've seldom seen enterprise architects design their software components and services to include a management interface in addition to whatever business interfaces are needed. There is no comprehensive offering from the vendor community yet, although management platforms are starting to emerge. Many enterprises are creating reusable application frameworks and then using them to build their applications. If inclusion of a management interface to software components and services is instilled into the enterprise architects, it will ensure that the applications are now clean, organized, well integrated and well managed.

As enterprises embrace Web services and SOAs and transform themselves into service-oriented enterprises, the road ahead starts with renovating the basement and utilizing all the available features of the Java platform.

**Venkat** heads technology consulting at Infosys for the mid-west region. He has over 12 years of experience in distributed computing, technology strategy and enterprise architectures. He served as the chief technology officer in a software company prior to joining Infosys, developing middleware and embedded systems products. Venkat has completed PhD courses in interdisciplinary studies and has an MS in aerospace engineering from the University of Alabama, Tuscaloosa, and a B.Tech from The Indian Institute of Technology, India.

*venkat@sys-con.com*

SYS-CON MEDIA

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

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

**Editorial**
Executive Editor:
**Nancy Valentine** *nancy@sys-con.com*
Associate Editors:
**Jamie Matusow** *jamie@sys-con.com*
**Gail Schultz** *gail@sys-con.com*
Assistant Editor:
**Natalie Charters** *natalie@sys-con.com*
Online Editor:
**Martin Wezdecki** *martin@sys-con.com*

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

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

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

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

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

Don't take chances connecting your application to your data. Rely on DataDirect Technologies™ for premium JDBC drivers with support for advanced functionality like distributed transactions, connection pooling, and BLOB/CLOB. Our Type 4 drivers are fully database independent and are the SPECjAppServer/ECPerf performance and scalability leader.

DataDirect™ TECHNOLOGIES

**Download your free evaluation today.** www.datadirect.com/jdj

www.datadirect.com 800-876-3101

**Yakov Fain**
Contributing Editor

# Java on **Wall Street**

Front office financial applications that place and execute orders are different from many others, since real-time trading systems must be blazingly fast and reliable. A few seconds delay may cost a financial brokerage company millions of dollars and potential penalties.

If, back in the '90s, you'd suggested using Java for processing split-second stock market orders, most of the New York programmers would just simply say: "faggedaboudid." If you want to have some fun, read old articles on using Java for Wall Street applications. Here are some statements made a long time ago in 1997:

- Java is strong on the front end, but we do not foresee it being used for very large number-crunching applications.
- Java is fine only for very thin clients.

A couple of years ago I was participating in the design and development of a multitier and multiplatform equities trading system that was built around Java Messaging and Enterprise JavaBeans (both session and entity beans). This real-time system has successfully replaced a legacy C++ application, has been deployed in production, and worked happily ever after. Not only was it more stable than the legacy system, but it was a lot more scalable. Multithreaded listeners retrieve orders from one or more message queues and send them to a cluster of J2EE application servers. Need more processing power? Just add another application server to the cluster, add more queues, and purchase additional communication lines with the stock exchange. No code changes required.

In September I attended the conference "High Performance Technology on Wall Street." If I had to describe this event in only one word, I'd use the word "grid." If I was allowed to add a second word, this would be "blades," and the third one would be "Java."

Software vendors have casually talked about using various Java technologies in high-speed real-time applications. Most of the vendors were either presenting software or hardware for grid computing. Blade servers are also becoming popular. Blades have nothing to do with shaving. Just imagine a metal cabinet with multiple narrow slots. Each slot hosts a blade server, which is a board with two or four processors, memory, and a local hard drive. All blades share high-speed I/O switches for communication with the rest of the world. Grid servers and agents are the software that supports such parallel computing.

Speakers presented colorful diagrams with hundreds of parallel jobs running on a grid; if one of the servers fails, the job gets redirected to another blade. Nice! But let's look at this technology from a practical point of view. Proprietary computation-centric, financial analytic software can be more or less easily divided into a set of parallel Java jobs. But how about running a hundred parallel application servers? This is also possible…if you have the budget to purchase hundreds of licenses for production, contingency, and QA environments. Most likely, these hundreds of servers will need to access either some data warehouse or a transactional database. If your system can't move the data fast enough between your application servers and the database, I/O may become a bottleneck of such a system. It's like driving a Ferrari on local streets.

I like this powerful technology and encourage you to present it as an option to your users. Can you process terabytes of data? Yes! Can you double the throughput? Sure! Technology is available…as long as you guys can afford it.

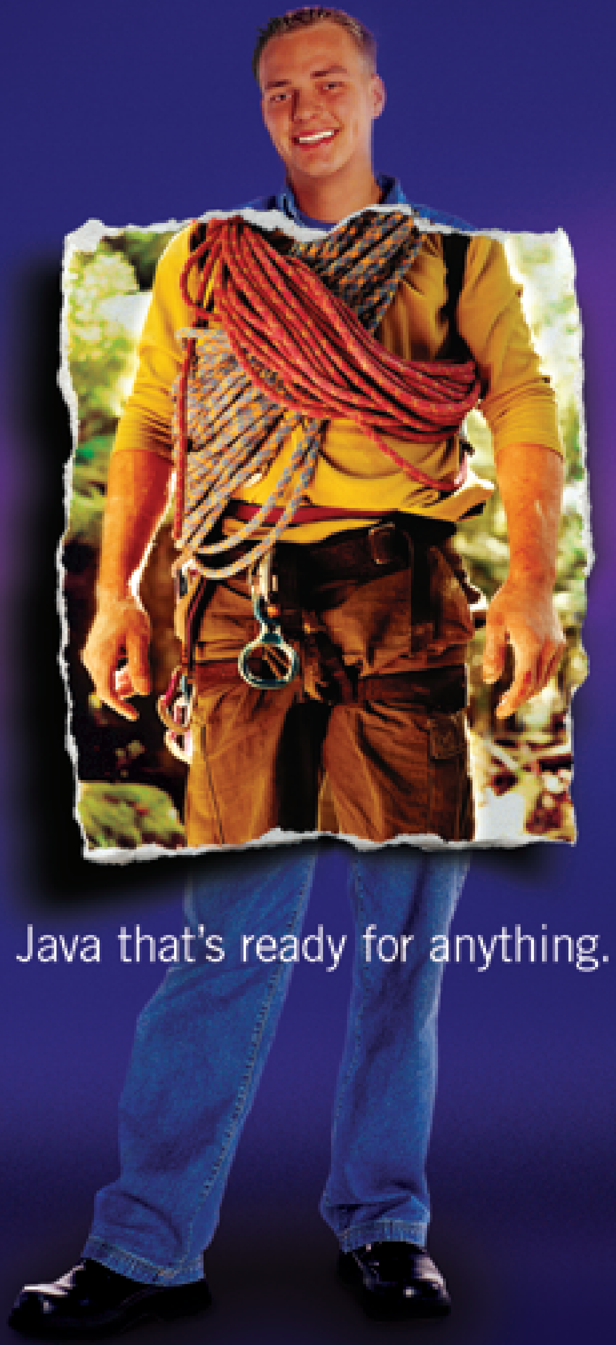
Another interesting topic was using XML for real-time systems. We already got used to application servers, database servers, Web servers, directory servers, intelligent business servers…please welcome: XML Server Farm. These agricultural machines are responsible for parallel XML parsing. If a system needs to send an order to a stock exchange to buy a hundred shares of SUN, we try to minimize the number of bytes that have to be processed, and "SUN,100" looks more attractive than "<Symbol>SUN</Symbol><Quantity>100</Quantity>". Who knows, maybe a couple of years from now the "slow XML" will be as funny as a "slow Java" is today.

Java feels at home in middle and back office applications that calculate risk and perform financial modeling utilizing functions for nonlinear optimization, statistical analysis, time-series analysis, and others. Some applications analyze trades that have already happened. As the brokerage industry introduces more and more regulations, financial giants are being fined heavily for cutting corners and breaking the rules. Applications that can process enormous amounts of data and weed out violations receive prime funding. Even though these Java applications may not need to process orders in real time, they also need a lot of power to sift the terabytes of data through various rule engines. These business intelligence servers use such in-memory gadgets as embedded Java databases, asynchronous nonpersistent queues, data caching, and parallel processing. Have I mentioned grids and blades yet?

Some heavy-duty Java gurus try to stay away from business applications, believing that the real fun coding is in companies that develop compilers, browsers, search engines, application servers, and the like. Trust me, these IT guys on Wall Street are not counting crows either. What's even more important for real geeks, you can work for a solid financial company and have as many earrings as you'd like, a long ponytail, grow a beard, and wear T-shirts and jeans. Wall Street welcomes the James Gosling look and feel!

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

*yakovfain@sys-con.com*



**Borland® JBuilder® 2005,** the newest edition of the #1 Java™ IDE in the world. With the most advanced feature set ever, ideal for enterprise-class projects. Complete command and control for JavaServer™ Faces (JSF™), JavaServer™ Pages (JSP™), servlets, Enterprise JavaBeans™ (EJB™), Web Services, Struts, XML, Swing, database applications, mobile applications, and more. Take your next project up to the next level. And enjoy peak performance.

**• Import project source from any IDE or editor • JSF-enables existing Web applications • New distributed refactoring for better teamwork • Customizable code editor with CodeInsight™ and ErrorInsight™ • Two-way visual Struts designer • Supports Apache™ Axis 1.2, SOAP, WSDL, UDDI, and WSIL • JSP™ tag library/framework support • Tight integration with Borland® Enterprise Server, BEA WebLogic™ Server, IBM® WebSphere,® and more**

**go.borland.com/j1**

**Borland®**

# Understanding Portals and Portlets

## Creating a customized portal

by Ken Ramirez

It used to be difficult if you wanted to create a Web-based site that offered users the ability to access various systems from a single page. Systems were too severely disjointed and required a huge investment of time and work in order to bring them together in a single Web page.

Although there are many efforts taking place in the Java arena to provide systems integration, none have made the same impact as portals.

### Understanding the JSR 168 Specification

The current breed of portals either already support the JSR 168 specification, or the developers are working quickly to try and bring them up to par with the specification. JSR 168 provides instructions to both portal software manufacturers and portlet developers.

For portal software manufacturers, this JSR provides the details on how a portal should be developed, what interfaces and objects need to exist, the communication process and sequence that should occur, security, and much more. For the portlet developer, it provides the interfaces that need to be implemented and used, the lifetime and scope of the portlets, and other related details.

There is yet a third perspective in this whole story, that of the end user. The end user views a portal as a collection of portlets presented on a single portal page, produced from requests made to a portal site (see Figure 1).

As an example, if you surf over to http://my.yahoo.com, you can register to have a personalized page that presents you with the content you sign up for, all from a single portal page. As shown in Figure 2, the page provided me with a Message Center Portlet, an RSS Headlines Portlet, a Scoreboard Portlet, and several other portlets that I had signed up for.

**Ken Ramirez** has 17 years of experience providing development services, consulting, and training to companies (both large and small) throughout the United States. He consults in various market industries including finance, insurance, computer-aided design, community portals, and automobile. Ken's Java expertise includes J2EE, XML, portals, UML, and many open source technologies. His latest venture is the www.TheJavaThinkTank.org community portal site.

*kramirez@TheJavaThinkTank.com*

Before continuing, let's make sure you understand the concepts and terms.

- ***Portals:*** An HTTP-based site hosted with special portal software that allows the aggregation of several different back-end systems, processes, or sites brought together through a single portal page. Portals may provide additional services such as single sign-on security, customization, personalization, and back-end administrative/declarative application development.
- ***Content aggregation:*** The process of bringing together content from disjointed systems, via portlets, and controlled through the use of a portal.
- ***Portlet container:*** Controls the access, lifetime, and interaction of a single portlet. Provides the content returned from a portlet back to the portal for merging with the content of other portlets.
- ***Portlet:*** Provides content to its calling portal container for the purposes of being displayed on a portal page.
- ***Fragments:*** The content generated by a portlet is known as its fragment or fragment code. This is the HTML code generated from the portlet's rendering code.

Figure 3 depicts the relationship among the entities specified above.

Since a portlet provides the rendering code for its portlet window, it's responsible for providing the implementation. However, most portlets will dispatch and rely on a JSP to provide the actual rendering code. Figure 4 shows the sequence as it applies to a portal page that has just received an action in one portlet. That portlet, along with the other portlets on the page, must also render themselves.

### Servlets and Portlets

Portlets share many similarities with servlets except for a few noted items. Portlets don't generate complete HTML documents; they're only interested in generating fragments that are included on the final portal page. They aren't allowed to generate HTML code that contains tags such as base, body, iframe, frame, frameset, head, HTML, or title. Imagine what would happen if all the portlets decided to provide a body tag. The portal wouldn't know whose body tag to use. The portal decides where these tags should go and provides additional tables, rows, and columns as needed for each of the portlets.

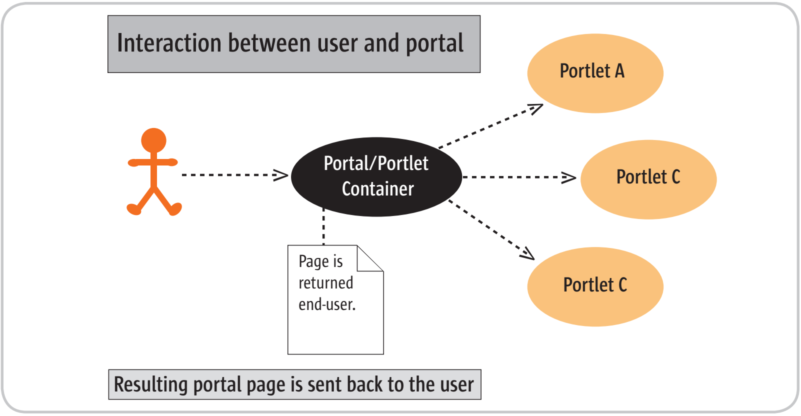

Interaction between user and portal
Portal/Portlet Container
Portlet A
Portlet C
Portlet C
Page is returned end-user.
Resulting portal page is sent back to the user

**Figure 1** Interaction between the user and the portal

Portlets aren't directly tied to a particular URL. Instead, they use methods such as createActionURL() or createRenderURL() to construct a URL that is needed to allow a client to fire actions to retrieve renderings from the currently executing portlet. On the client side, clients aren't allowed to interact directly with a portlet. The requests or submissions are tunneled through the portal server. The URL has all the information that the portlet container needs to determine which portlet must be called and what type of functionality should be executed.

Two other very important differences are elaborately pronounced in the user interface. Anyone viewing a portal page for the first time will notice that they contain special adornments that can be utilized by users to minimize, normalize, or maximize the portlet window. Users also use the decorations to enter the Edit mode or View mode of the portlet (see Figure 5). Finally, portlets can exist on the same page multiple times. Most of the time, the user is given the ability to control which portlets show up on a particular portal page. This is known as personalization.

Although portlets can access both servlets or independent JSPs directly by including their output within the portlet's rendered output, the direct output of a servlet or an independent JSP should not be channeled back to a portal page unless the content is stripped of all of the offending HTML tags mentioned above. Dispatching is handled through a special object known as a PortletRequestDispatcher. Besides passing control directly to another portlet, servlet, or JSP, the portlet may choose to simply include the output from the entity, without losing the ability to provide further output appended to the end, essentially becoming an aggregator.

## The Life of a Portlet

A portlet is initially called by the portlet container. In fact, its entire life is managed by the container (including requests and responses). The portlet inherits from the GenericPortlet class. The container loads the portlet and then calls its init() method. There are two versions of this method. The first version receives the PortletConfig object and stores it away for later use. This version then calls the parameterless version to perform any overridden code provided by the portlet author. Keep in mind that the portlet may be loaded and its init() method may have been called, but it might not be provided with a session until later. This is important to note, because if you need to store attributes within the init() method, you'll need to use the context object.

Within the deployment file that accompanies the portlet distribution file, the author can place initialization parameters, which can be pulled out by the portlet instances at runtime via the configuration object. This object can be pulled out by calling the GenericPortlet's getPortletConfig() method, which returns a PortletConfig object. Using this object, the author can access the initialization parameters by calling the object's getInitParameter() for each parameter that the portlet is interested in.

Portlets can also place titles and other pieces of information in accordance with the XML Schema into the deployment file. In addition, any displayable information can be kept in a resource bundle associated with the deployment descriptor file and be accessible at runtime. The values can be language-specific to assist in internationalizing the portlet.

Each time the client performs an action on a portlet, the portlet is invoked by the container via its processAction() method. When the portlet container finds it necessary to retrieve a viewing of the portlet, it calls the render() method. This method determines the type of view that should be rendered and calls one of three other methods: doHelp(), doView(), or doEdit(). These three methods should be overridden by the portlet author if the author expects to receive calls for any of these methods. The calling of these methods is controlled by the portlet author through the portlet deployment descriptors file (portlet.xml).

When the portlet container calls the action method of the portlet, there are two objects passed to it: first ActionRequest and, second, ActionResponse. Using the ActionRequest, the portlet can access the parameters of the request, the window state, the portlet mode, the portlet context, the session object, and the portlet preferences data.

The same two objects are sent to the render method of the portlet, except with slightly different names: RenderRequest and RenderResponse. Keep in mind that if you wish the parameters to become available in the render method or the resulting JSP, you'll need to move it there yourself using the ActionRe-

Figure 2 An example of a portal page

sponse.renderParameters() method, passing it the result of calling ActionRequest.getParameterMap(). You can later fetch the parameters from within the render method using RenderRequest.getParameter().

Remember the relationship between the URL used to invoke the portlet and the actual method that is called within the portlet. If you expect that an interaction from the user will result in a call to the portlet's processAction() method, you should call createActionURL() to have that type of URL created. For render requests, call createRenderURL().

When the portlet has completed its job, the container will call the portlet's destroy() method.

## Portlet Preferences

Users expect that portlets can be modified to provide customized content. To provide customization, portlets support preferences, which are name/value pairs that can be assigned an initial value and later tailored to other values based on user preference.

Preferences can be modified during the processAction()method and inspected during any of the render methods. Preferences are altered or viewed through the PortletPreferences object. The interface provides methods to retrieve, change, or store preferences. Since preferences can be read-only, there is also a method to check if a preference can be modified.

Changes to preferences aren't committed until the store() method is called. It may only be called during the processAction() method.

Preferences are initially defined within the portlet's deployment file and given a default value. In addition, developers may assign a validator object, which implements the PreferencesValidator interface. This object's implementation can verify that values assigned to preferences are legitimate. Otherwise, the validator can throw a ValidatorException. Developers are encouraged to include information in the exception regarding the preferences that failed.

## Portlet Modes and Window States

The portlet modes of a portlet are reflective of the kind of data the current portlet is displaying to the user. The standard modes are View, Edit, and Help. In the View mode, the portlet should display the normal content, based on the functionality offered by the portlet. For example, if the portlet is a stock portfolio portlet, it should display the symbols and current price/change for this mode.

In Edit mode, the portlet should present the user with the ability to customize whatever features are customizable for the portlet. For our stock portfolio portlet example, the portlet might present the ability to adjust the stocks in the portfolio, changing the colors to show up and/or down activity or changing the background color for the portlet.

The Help mode should display either a full description of how the portlet can be used (explaining both the View and Edit modes) or context-sensitive help explaining selected features of the portlet.

If the portlet wishes to determine which mode it's currently in when called, it can do so by calling the ActionRequest or RenderRequest objects' getPortletMode() method.

Portlets should specify their desire to handle any of the portlet modes by including the modes within the deployment descriptor's <supports> element.

The render() method will fall back to one of the three methods – doView(), doEdit(), and doHelp() – in order to have the portlet provide the implementation, depending on what portlet mode is currently in use. Most portal pages will provide the functionality that enables the users to choose which mode they wish to enter. This can be done by adornments placed on the title bar of the portlet.

The Window states allow the portlet to know the user's aspiration to have the portlet minimize its window, maximize its window, or normalize the window (bringing it from minimized or maximized mode). The portlet should always check what Window mode it's in before rendering data.

## Portal Context and Session

The portlet can retrieve context information about the portal hosting it by accessing the PortalContext object. Utilizing this object, the portlet can call getPortalInfo() to retrieve vital information such as the portal vendor and portal version. It can also retrieve portal properties utilizing getProperty() and getPropertyNames(). The PortalContext object can be retrieved by calling the getPortalContext() method.

Portlets can store session data associated with the user's active session. There are several ways in which a server process can store data for later use within the session. These include:

- HTTP cookies
- SSL sessions
- URL rewriting

The portlet specification provides the developer with an interface named PortletSession that hides the implementation details, providing a simple interface for managing the various data pieces in a session utilizing the above-mentioned methods. The portlet session is guaranteed to be the same session across all portlets that result from requests made by the user.

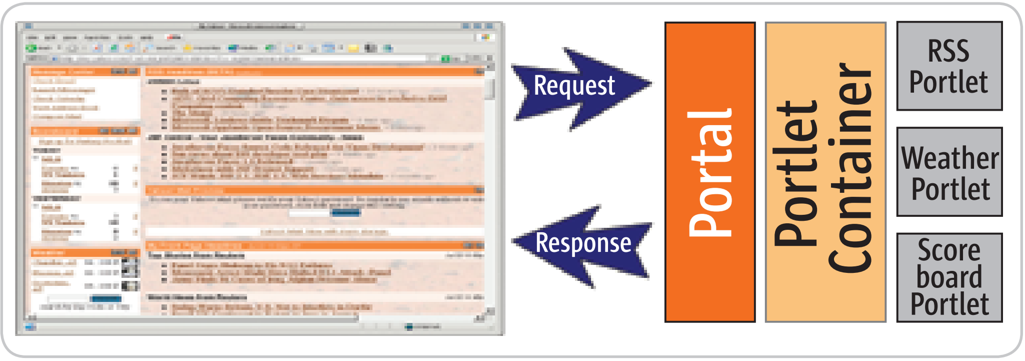


**Figure 3** Relationship between the entities

**Now your ideas can get even bigger.**

**Visual Studio .NET 2003 significantly reduces the amount of code you need to write, freeing you up to think big thoughts.**

Think big. Visual Studio® .NET 2003 delivers higher productivity by taking care of the small stuff, so you can concentrate on bigger things. Just ask HP. "On our consumer application project Visual Studio .NET really lowered the technical barriers to doing great work," says an HP Technical Lead. "We developed the application faster than we could have with other technologies due to the better tool set capabilities." To find out how Visual Studio .NET 2003 can help you concentrate on big ideas, visit msdn.microsoft.com/visual/think

Microsoft® Visual Studio®

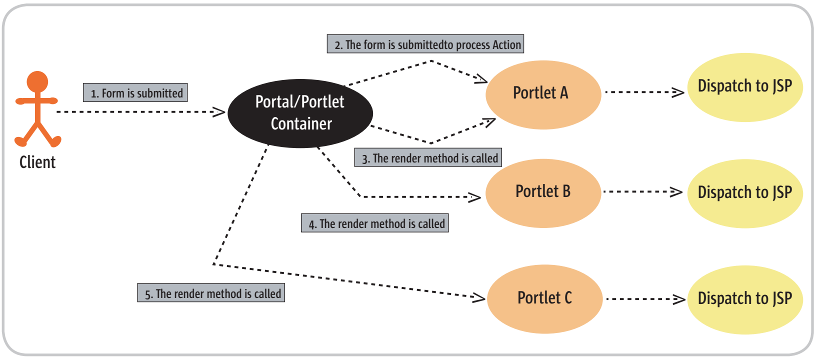

Client
1. Form is submitted
Portal/Portlet Container
2. The form is submittedto process Action
Portlet A
Dispatch to JSP
3. The render method is called
Portlet B
Dispatch to JSP
4. The render method is called
5. The render method is called
Portlet C
Dispatch to JSP

**Figure 4** Sequence of performing action and rendering

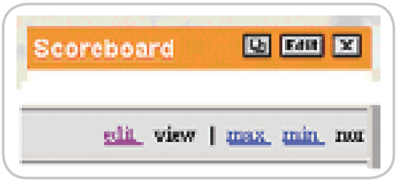


**Figure 5** Examples of some adornments

## Caching

Portals can provide portlets with a cache to store rendered output. The cache can be set to expire at a particular time, such as every 500 seconds. When required, each cache is created one to one with the user session and only for that particular portlet. The uniqueness of the cache is based on both the portlet and the user. The settings are placed in the portlet's deployment descriptor file distributed with the portlet. However, the portlet can alter the settings at runtime utilizing the RenderReponse object.

The portlet container can choose to use the requested portlet cache if resources permit; otherwise, it can shut down caching altogether. If the portlet container chooses to use caching, any request from the client to simply render content causes the portlet container to first check if the content has expired and, if it hasn't, returns the contents without calling any of the portlet's render methods. Otherwise, the methods are called once again, and the content is cached for later requests.

## Putting the Pieces Together

The true advantage to producing portlets is realized when content providers are able to integrate existing portlets onto a single page, essentially providing users with portal pages that result in a portal site.

The actual semantics for achieving this will differ from one portal vendor to another. For example, some vendors will provide an administrative tool that can be accessed to build the portal site, bringing together the portlets by allowing administrators/developers to search for the type of portlets they wish to offer users, and then providing the plumbing to make this all happen.

The end result of all the administrative effort will be a portal page deployment file or perhaps some kind of registry (which can even be handled via an LDAP system) to produce the resulting site. The portal then utilizes the information stored in these data stores to produce the site at runtime. The data stores might have information such as:

- The name of the portal class and its logical name
- A description of the portal class
- Preferred real estate location and size
- Roles that may view the portlet

The portal may also maintain information about the individual pages, such as:

- Which portlets are included in what pages
- Initial portlet modes or Window states
- Flow of one portal page to another

These are just some of the pieces of information that a portal may potentially store as the site is administered.

## Handle Your Threads

The portlet container handles multiple requests to a portlet by reusing the same instances of a portlet class, calling them on different threads through the same methods. This means that your code should be thread safe but yet avoid locking methods that can degrade performance or, worst yet, cause a deadlock. How can you provide a fast and efficient thread-safe method and yet still avoid locks? Make your portlets stateless. Sometimes object locks are unavoidable, but if they're necessary, try to do it with code that performs well and has the least chance of causing deadlocks.

## What Does the Future Hold?

Although the portlet specification is in final release, this is simply version 1.0 of this specification. Future versions of this specification will outline the plans for:

- ***Filters:*** Similar to servlet filters – in fact, some vendors have already implemented this optional feature using the same type of method offered by servlets.
- ***Interportlet communication:*** This will allow one portlet to hook itself into another portlet, so that it can immediately be alerted to changes to the second portlet.
- ***Outer markup modification:*** Today portlets can only produce markup kept within their fragments. Future versions of the specification will allow portlets to influence the markup outside of their fragments (perhaps influencing the background colors of other portlets based on changes to the preferences of one portlet).

## Summary

The information presented in this article was based solely on the portlet specification in its final release form. However, the only true way to learn any technology is to start using it. If you visit the Apache.org Web site (www.apache.org), you'll find the portals project. You should download and install the Pluto project and begin to generate some portlets. Pluto is the reference implementation for the portlets technologies and, as such, is completely compliant with the latest and greatest iteration of the portlet specification.

Next month, I'll be back to show you how to build an Image Viewer Portlet that will allow you to select images from the Web site you wish to view. In the process, you'll become acquainted with many of JSR 168's features, and also learn how to deploy portlets to Pluto.

ALTOVA® stylevision® 2005

# Focus on results

**Set your sights on StyleVision® 2005, the ultimate e-forms, DB report and stylesheet designer from Altova®.**

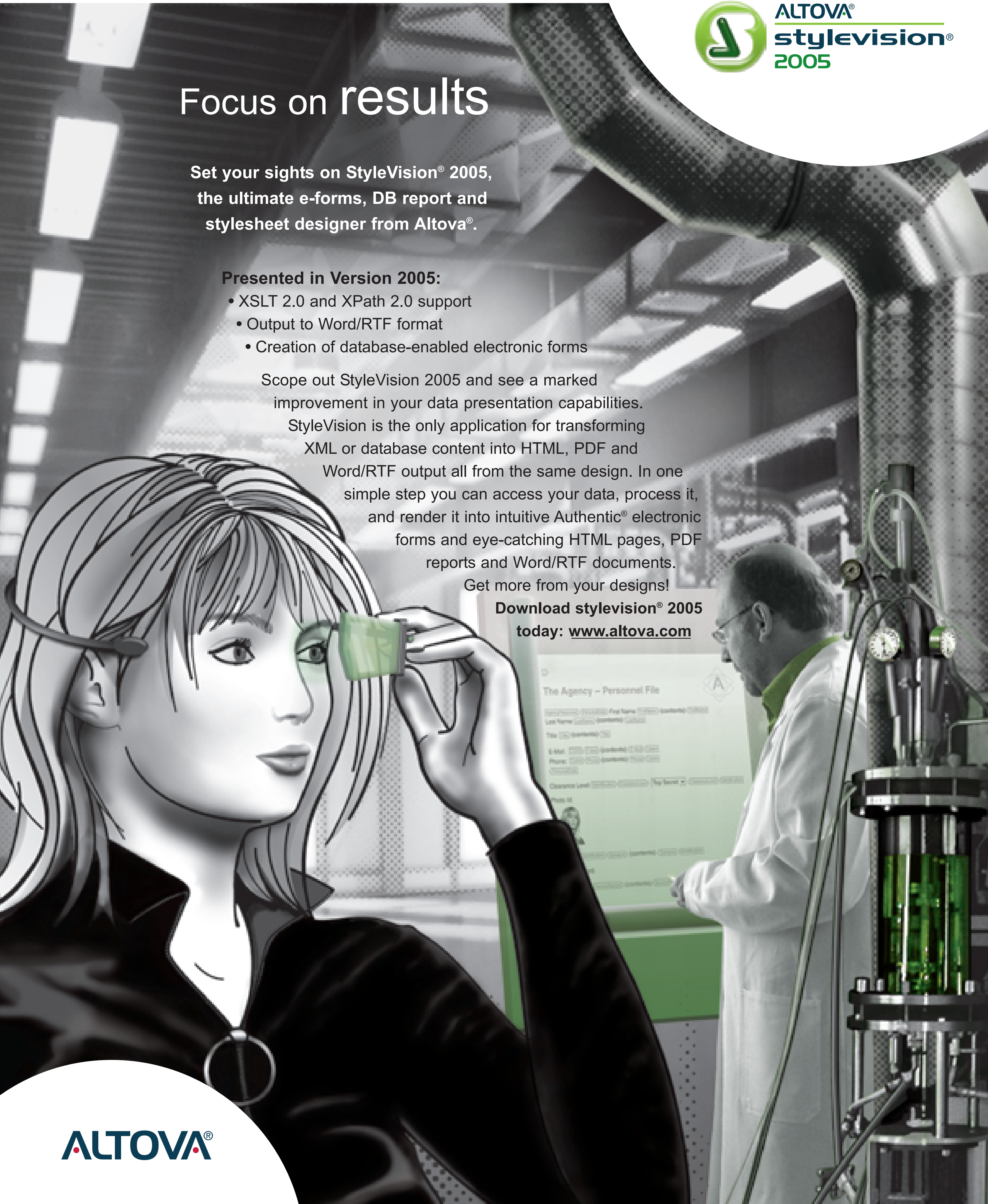
**Presented in Version 2005:**

- XSLT 2.0 and XPath 2.0 support
- Output to Word/RTF format
- Creation of database-enabled electronic forms

Scope out StyleVision 2005 and see a marked improvement in your data presentation capabilities. StyleVision is the only application for transforming XML or database content into HTML, PDF and Word/RTF output all from the same design. In one simple step you can access your data, process it, and render it into intuitive Authentic® electronic forms and eye-catching HTML pages, PDF reports and Word/RTF documents.

Get more from your designs!

**Download stylevision® 2005 today: www.altova.com**



ALTOVA®

# EJB 3.0 Preview

## Part 1: The basic programming model

by Bill Burke

This article is part one of a two-part series on the new Enterprise JavaBeans (EJB) 3.0 specification. Prior knowledge of J2EE/EJB will enable a better reading experience. Part 1 focuses on the basic programming model of EJB 3.0. Part 2 will focus on more advanced features like dependency injection and complex persistence mappings (entity inheritance and multitable mappings).

Over the past 15 years, each revision of middleware specifications like DCE, CORBA, and J2EE evolved into a larger, more complex definition of new functionality and bloatware. Rarely has a standards-based specification stepped back and actually tried to make development easier for its user base.

Until now that is. The mandate of the EJB 3.0 expert group to focus on ease of use and simplification is a refreshing unprecedented change in a standards body. This article focuses on the goals of EJB 3.0 and walks you through the new model for session and entity beans.

### The Difficulties of EJB 2.1

There are a lot of problems in EJB 2.1 that make it difficult to develop EJBs. XML deployment descriptors have continually been the bane of developers. Tools like XDoclet have helped alleviate some but not all of the complexity. Removing these files would go a long way toward making things simpler. One big annoyance in EJB is the sheer verbosity of the API. You have to create a number of interfaces and classes to implement one EJB: a remote and a local interface, a remote and local home interface, and finally a bean class. Your bean classes have to implement EJB interfaces that have numerous callback methods like ejbCreate or ejbPassivate that are often never even used at all as part of the application logic. Yet another complaint is that EJBs are completely untestable outside the context of the container as components like entity beans are abstract classes. Finally, EJBQL in its current form is so significantly hobbled that developers continually have to escape to straight JDBC and SQL, or ditch their CMP efforts entirely and replace them with something like Hibernate.

Bill Burke is chief architect of JBoss Inc., member of the EJB3 expert group, and co-author of the *JBoss 4.0 Workbook* in O'Reilly's Enterprise JavaBeans, 4th Edition.

*bill@jboss.org*

Fortunately, the EJB 3.0 specification is working to address most of these problems. Let's take a look at how this is being accomplished.

### Deployment Descriptors

JDK 1.5 has a new feature called annotations defined in the JSR-175 JCP specification. Annotations allow you to define configuration information as an interface, then apply instantiations of this configuration as metadata attached to a class, method, field, constructor, package, or parameter. For those familiar with XDoclet, it is very similar except the metatags are typed syntax that is checked by the Java compiler. This metadata is even available at runtime. The EJB 3.0 specification uses annotations so that you can declare your EJB metadata directly within the bean class.

```
import javax.ejb.*;

@Stateful
public class ShoppingCartBean implements
ShoppingCart
{
   @Tx(TxType.REQUIRED) @MethodPermission(
{"customer"})
   public void purchase(Product product,
int quantity) {...}


   @Remove void emptyCart() {...}
}
```

The @Stateful annotation marks the ShoppingCartBean as a stateful session bean. @Tx denotes transaction demarcation, while @MethodPermission defines role-based security for the bean method. EJB 3.0 provides annotations for every type of metadata so that no XML descriptor is needed and you can deploy your beans simply by deploying a plain old JAR into your application server. This doesn't mean that XML completely disappears; it becomes optional. If you don't like to expose infrastructure metadata directly in application logic, all annotation types are overridable in an XML deployment descriptor.

### Simplified API

The stateful session bean example above is complete. You'll notice that without the annotations, ShoppingCartBean is a plain old Java object (POJO). It does not have to extend javax.ejb.SessionBean, or implement any of the callback methods like ejbPassivate(), ejbCreate(), ejbRemove(), etc. Session beans must also implement at least one interface. What's great about this interface is that it, too, is plain Java.

```
public interface ShoppingCart
{
   public void purchase(Product product,
int quantity);
   public void emptyCart();
}
```

Another goal of EJB 3.0 is to provide common sense defaults. A session bean that implements only one interface treats that interface as a local interface by default. If you wish to make your EJB remote, you must explicitly tag the interface as such.

```
@Remote interface ShoppingCart
{
   public void purchase(Product product,
int quantity);
   public void emptyCart();
}
```

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

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

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

**// JOIN IN AT SDN.SAP.COM**

Remote interfaces do not have to extend javax.ejb.EJBObject, nor do any of the methods have to throw a RemoteException as in the EJB 2.1 specification. EJB 3.0 tries to remove the dependencies on RMI APIs.

## Callbacks a la Carte

The EJB 2.1 spec required you to implement the interfaces javax.ejb.SessionBean or javax.ejb.EntityBean. Methods like ejbCreate(), ejbPassivate(), and ejbActivate() were never used in your application and just cluttered up your code. In EJB 3.0 you can use these methods a la carte on an as-needed basis.

## Homeless

Home interfaces have been completely removed for all EJB types. They never made much sense for stateless beans, and had only limited use for stateful sessions. Looking up a session bean gives you a proxy that you can immediately use to invoke on the session. Every time a stateful bean is referenced from JNDI, a new instance is created. It is assumed that the first method called should initialize the bean. Methods marked as @Remove will destroy the stateful session after the annotated method has completed. Entity beans are created, removed, and queried through the new EntityManager interface which will be described later in this article.

## Entity Beans

Entity beans are by far the biggest change in EJB 3.0; they received a complete overhaul in this version of the specification. One thing that the EJB 3.0 expert group realized is that a one-size-fits-all approach to persistence does not produce a very usable API. For instance, the portability of 2.1 entity beans is nonexistent as different vendors have different database mappings. EJB 3.0 focuses on an object-to-relational (O/R) mapping that supports inheritance, multitable mappings, join-tables, a fully functional query language, and SQL escapes. These are all the things you would expect from an O/R persistence engine. Another important change to entity beans is that they are pure plain Java objects and can no longer be remotely accessed, or provide transaction or role-based security boundaries. This approach to a pure POJO has many advantages – the biggest being that you can detach and reattach entities to persistence storage. We'll see more on that later in this article.

Let's walk through Listing 1. As you can see, the order entity bean is a plain Java object. The @Table annotation specifies the table name. The class contains private fields that represent the state of the bean, while get and set methods wrap the access to these fields. The column mappings are applied as annotations on the get methods of the class. If you don't like this approach, you can declare these mappings on the fields. @Id is used to specify the primary key field. @Column specifies the column mapping while the @OneToOne and @OneToMany persistent fields use @JoinColumn to specify the foreign key column of the related table. OneToMany and ManyToMany relationships are specified as a Collection generic.

All in all, the O/R mapping was designed to be compact yet flexible with intuitive defaults. For instance, if you were relying on the container to do auto-generation of your database tables, only the @Entity and @Id annotations would be required as the rest of the persistence metadata would have well-known defaults.

## Interacting with Entity Beans

Entity beans no longer have homes. They are created as plain Java objects using Java's new operator and all interaction between persistence storage and application code is handled by a new service called the EntityManager. Listing 2 shows an example of this.

## Attachment, Detachment, and Reattachment

What is interesting about the checkout() method of our ShoppingCart bean is that it returns the order object that it created. In EJB 3.0, the old antipattern of value objects is not needed because persistent objects can be attached, detached, and reattached to persistence storage. For example, you could have a remote client that created all the OrderItems and the order object on the client-side, send the order object over the network in a remote EJB call, and have the EntityManager create the order, or if you're updating, *merge* the changes to the persistent object. Because they are plain Java objects, entity beans can be used in <jsp:useBean> tags and stored in an HTTP session. You can imagine a five-step wizard that allocates a group of entity beans that holds the state of the wizard, then, when the wizard completes, interacts with the EntityManager to create all the persistent data. The programming model becomes much simpler as you have one object that you can pass around from tier to tier to handle your creates and updates. Here's an example of doing updates remotely:

*Client*

```
   Session session = jndi.
lookup("Session");
   Customer cust = session.
getCustomerByName("Bill Burke");
   cust.setAddress("123 Boston Road");
   cust.setCity("Concord");
   cust.setState("MA");
   cust.setZip("02143");

   session.updateCustomerInfo(cust);
```

In the above example, the remote client gets access to a customer object through a method on a session bean. All the setter methods are done locally to an instance of customer in the local VM. When the client is finished updating the customer locally, it

"The mandate of the EJB 3.0 expert group to focus on ease of use and simplification is a refreshing unprecedented change in a standards body"

Nothing beats our racks

Absolutely nothing

**CARRIER CLASS DATA CENTERS**

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

Robert Marsh, Head Surfer

**START YOUR OWN WEB HOSTING BUSINESS TODAY!**

from **$299***

Instant Activation!

**Dedicated Server**

**Dual Xeon 2.4 GHz**
2 GB RAM • 2 x 73 GB SCSI HD
Remote Console • Remote Reboot
1700 GB Monthly Transfer Included

**Over 20,000 Servers!**

1-800-504-SURF | ev1servers.net

PLESK7 RELOADED
Preferred Control Panel

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

sends the customer over the network to the remote session bean to be updated in persistent storage. Listing 3 shows how the session bean would be implemented.

The getCustomerByName() method simply searches for a customer of a given name. When the customer object is returned, it is detached from the persistence engine and is no longer associated or managed by the EntityManager. The updateCustomerInfo() method receives the fully modified customer and reattaches the object by calling the merge() method. This causes the EntityManager to do an update on the customer's row in the database.

## Querying

Unlike the EJB 2.1 specification, 3.0 has full support for dynamic queries. You can build a query object through EntityManager.createQuery(), then interact with the query object to set page sizes and arguments.

```
   Query query = entityManager.
createQuery("from Order o where
o.grandTotal > 5000.00");
   query.setMaxResults(50);
   return query.listResults();
```

EJBQL is now a fully featured query language that mirrors the functionality of SQL. Support for group by, having, inner and outer join, subqueries, and bulk update and delete has been added. Also, queries can now return more than one value as well as a list of objects. One of my favorite features of the new query language is the ability to project results onto any arbitrary Java object by using a constructor directly in the query. For instance, say you wanted a page on a petstore application that displayed a report on sales divided by geographical area. The query might look something like this:

```
SELECT new GeographicalReport(c.state,
sum(o.grandTotal)) From Customer c
join Order.customer Customer
GROUP BY c.state
```

GeographicalReport is not an entity bean, but rather a plain Java object. The entity manager would execute the query and allocate a GeographicalReport object per row returned in this example. Instead of iterating though a potentially large untyped result set, you can have the query manager automatically populate the data structures you need.

## Coming Soon

The first draft of the EJB 3.0 specification was announced in June at JavaOne. As is, it is a good first step at simplifying the EJB programming model and fixing some of the deficiencies in the persistence model. Although the specification is not due to be finished until next year, some vendors have committed to providing early-access downloads so that you can play with this new technology. All in all, these are exciting times for EJB.

Next month we will dive into more advanced features of EJB 3.0.

## References

- www.jcp.org/en/jsr/detail?id=220
- www.jboss.org/ejb3

**Listing 1**

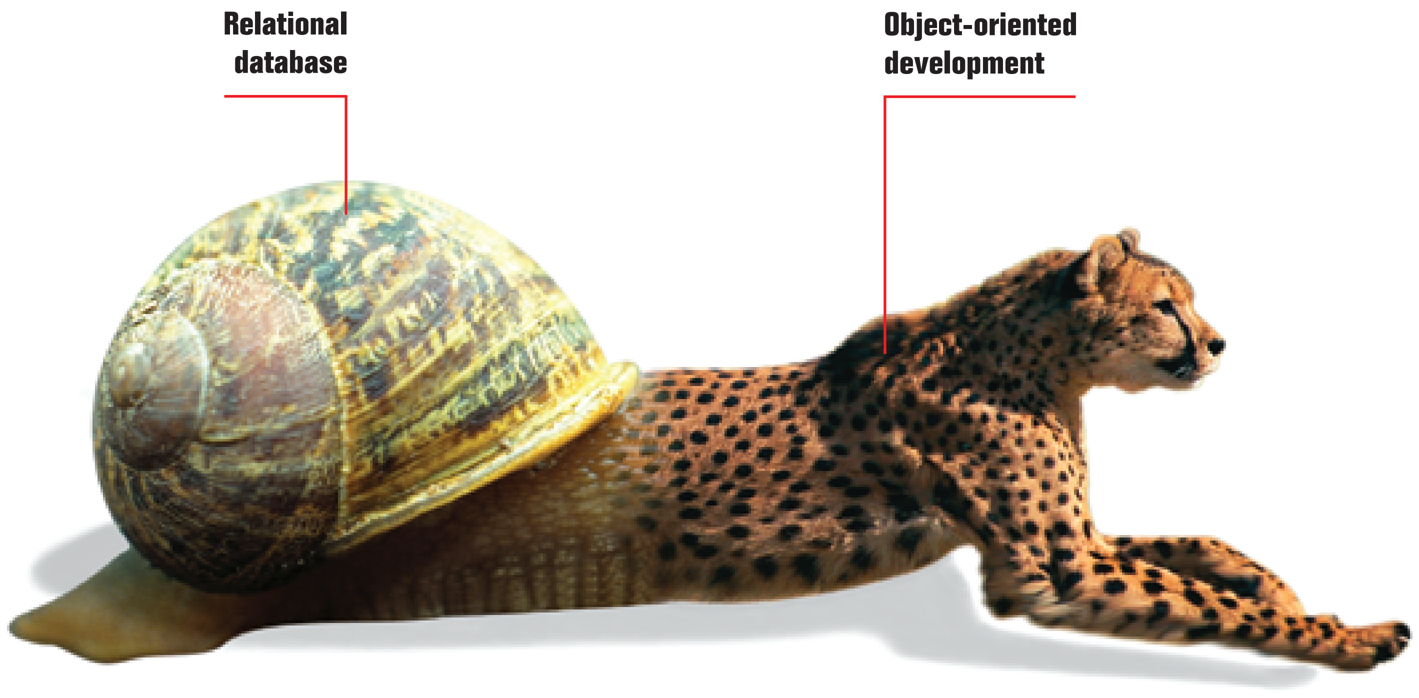
```
@Entity
@Table(name="ORDER_TABLE")
public class Order implement java.io.Serializable
{
   private int orderId;
   private Date orderDate;
   private Collection<OrderItem> orderItems;


   @Id @Column(name="ORDER_ID")
   public int getOrderId() { return orderId; }

   public void setOrderId(int id) { orderId = id; }

   @OneToOne @JoinColumn("CUST_ID")
   public Customer getCustomer() { return customer; }

   public void setCustomer(Customer cust) { this.customer = cust;
}

   @OneToMany(cascade={CascadeType.ALL}) @JoinColumn("ITEM_ORDER_
ID")
   public Collection<OrderItem> getOrderItems() { return
orderItems; }

   public void setOrderItems(Collection<OrderItem> items) { this.
orderItems = items; }

}
```

**Listing 2**

```
@Stateful
public class ShoppingCartBean implements ShoppingCart
{
   @Inject EntityManager entityManager;
...

   public Order checkout()
   {
      Order order = new Order();
      order.setCustomer(this.customer);
      order.setOrderDate(new Date());
      order.setOrderItems(this.itemsInCart);
      order.setOrderStatus("INITIAL");

      entityManager.create(order);
      return Order;
   }
}
```

**Listing 3**

```
@Stateless
public class SessionBean implements Session
{
   @Inject EntityManager manager;

   public Customer getCustomerByName(String name)
   {
      Query query = manager.createQuery("from Customer c where
c.name = :name");
      query.setArgument("name", name);
      return query.getUniqueResult();
   }

   public void updateCustomerInfo(Customer cust)
   {
      manager.merge(cust);
   }
}
```



# DON'T LET YOUR DBMS SLOW YOU DOWN

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

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

That means huge savings in both development and processing time. Applications built on Caché are massively scalable and lightning fast. They require little or no database administration. And Caché's powerful Web application development environment dramatically reduces the time to build and modify applications.

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

InterSystems CACHÉ™
Make Applications Faster

**Try a better database. For free.**

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

# Creating a Pet Store Application

*...with JavaServer Faces, Spring, and Hibernate*

by Derek Yang Shen

JavaServer Faces (JSF) technology is a new user interface framework for J2EE applications. This article uses the familiar Pet Store application to demonstrate how to build a real-world Web application using JSF, the Spring Framework, and Hibernate. Since JSF is a new technology, this article will concentrate on the use of JSF. It presents several advanced features in JSF development, including Tiles integration and business logic–tier integration.

## Java Pet Store

The Java Pet Store is a sample application from the Java Enterprise BluePrints program. It documents best practices, design patterns, and architectural ideas for J2EE applications.

MyPetStore, the sample application for this article, is a reimplementation of the Java Pet Store using JSF, Spring, and Hibernate.

I won't be able to cover all the features of the Pet Store in one article. MyPetStore allows a user to browse through a catalog and purchase pets using a shopping cart. Figure 1 provides the page-flow diagram.

## JSF

JSF is a server-side, user interface component framework for J2EE applications. JSF contains an API that represents UI components and manages their states; handles events, server-side validation, and data conversion; defines page navigation; supports internationalization; and provides extensibility for all these features. It also contains two JSP (JavaServer Pages) custom tag libraries, HTML and Core, for expressing UI components within a JSP page and wiring components to server-side objects.

JSF is not just another Web framework. It's particularly suited, by design, for use with applications based on the MVC (Model-View-Controller) architecture. The Swing-like object-oriented Web application development, the bean management facility, an extensible UI component model, the flexible rendering model, and the extensible conversion and validation model are the unique features that differentiate JSF from other Web frameworks.

Despite its strength, JSF is not mature at its current stage. Components, converters, and validators that ship with JSF are basic. The per-component validation model cannot handle many-to-many validation between components and validators. In addition, JSF custom tags cannot integrate with JSTL (JSP Standard Tag Library) seamlessly.

## High-Level Architecture

MyPetStore uses a multitiered nondistributed architecture. For a multitiered architecture, the functionalities of an application are partitioned into different tiers, e.g., presentation, business logic, integration, etc. Well-defined interfaces isolate each tier's responsibility. A nondistributed architecture means that all the tiers are physically located in the same application server. Figure 2 shows you the high-level architecture of MyPetStore.

JSF is used in the presentation tier to collect and validate user input, present data, control page navigation, and delegate user input to the business-logic tier. Tiles is used to manage the layout of the application.

Spring is used to implement the business-logic tier. The architectural basis of Spring is an Inversion of Control (IOC) container based around the use of JavaBean properties. Spring is a layered application framework that can be leveraged at many levels. It contains a set of loosely coupled subframeworks. The use of the bean factory, application context, declarative transaction management, and Hibernate integration are demonstrated in this application.

The integration tier is implemented with the open source O/R (object/relational) mapping framework – Hibernate. Hibernate relieves us of low-level JDBC coding. It's less invasive than other O/R mapping frameworks, such as JDO and CocoBase. Rather than utilize bytecode processing or code generation, Hibernate uses runtime reflection to determine the persistent properties of a class. The objects to be persisted are defined in a mapping document, which describes persistent fields and associations, as well as any subclasses or proxies of the persistent object. The compilation of the mapping documents and SQL generation occurs at system startup time.

The combination of the business logic tier and the integration tier can also be referred to as the middle tier.

The integration between different tiers is not a trivial task. MyPetStore demonstrates how to use the JSF bean management facility and ServiceLocator pattern to integrate JSF with the business logic tier. By using Spring, the business logic tier and integration tier can be wired up easily.

**Derek Yang Shen** is a senior software engineer at Overture Services, Inc., in Pasadena, California. Derek is a Sun Certified Enterprise Architect and has been working with J2EE exclusively for the past five years. He holds an MS in computer science from UCLA.

*derek@derekshen.com*

## Implementation

Now, let's go through the implementation details, tier by tier, of the UpdateShoppingCart, the most important and complex use case in this application.

### *Presentation Tier*

The presentation tier tasks include creating and registering the backing beans (explained later), writing JSP pages, defining navigation rules, integrating with Tiles, and integrating with the middle tier. Our shopping cart screen looks like Figure 3.

### *Backing Bean*

Backing bean is a JSF-specific term. A backing bean defines the properties and handling logic associated with the UI components used on a JSF page. Each backing bean property is bound to either a component instance or its value. A backing bean also defines a set of methods that performs functions for the component.

Let's create a backing bean – CartBean – that contains not only the properties maps to the data for the UI components on the page, but also three actions: addItemAction, removeItemAction, and updateAction. Because the JSF bean management facility is based on Java reflection, our backing bean does not need to implement any interface. Listing 1 provides the code segment of the CartBean.

The CartBean contains a reference to a Cart business object. The Cart business object contains all the shopping cart–related data and business logic (the Cart class will be discussed later). This approach, to include the business object directly inside the backing bean, is simple and efficient. However, it tightly couples the backing bean with the back-end business object. Another approach is to decouple the backing bean and the business object. The drawback of this approach is that mapping has to be performed between the objects. Data needs to be copied between the backing bean and the business object.

There's no business logic inside the backing bean actions. The backing bean action simply delegates the user request to the middle tier. The addItemAction takes the item ID from the request, then it looks up the CatalogService through the ServiceLocator and gets the item associated with the item ID. It calls the addItem method on the Cart business object. The business logic of how to add an item to the cart is handled by the Cart business object. Finally, if everything succeeds, the navigation result of success is returned to the JSF implementation.

### *Backing Bean Registration*

To let the JSF bean management facility manage your backing bean, the CartBean must be registered in the JSF configuration resource file faces-managed-beans.xml (see Listing 2).

The CartBean is set to have a scope of a session, which means the JSF implementation creates a CartBean instance if it is referenced inside any JSP page for the first time during a session. The CartBean instance is kept under the session scope. This way the user can interact with the stateful shopping cart, add an item, remove an item, update the cart, and finally check out.

### *The JSP Page*

The cart.jsp is the page to present the content of a shopping cart. It contains UI components and wires the components to the CartBean (see Listing 3).

The page starts out with the tag library declarations. The JSF implementation defines two sets of tags. The core tags are independent of the rendering technology and are defined under prefix f. The HTML tags generate HTML-specific markup and are defined under prefix h. All JSF tags should be contained inside an f:view or f:subview tag.

h:outputText is used to present a message to the user once the shopping cart is empty:

```
<h:outputText value="Your Shopping Cart is Empty"
  styleClass="title" rendered="#{cartBean.numberOfItems <= 0}"/>
```

The rendered attribute takes a boolean variable. If the value is false, the h:outputText will not be rendered. The rendered attribute can give you the same effect as a JSTL c:if. Since JSF and JSTL are not integrated well, it's good practice to try not to mix them together.

h:dataTable is used to iterate through the items inside the shopping cart and present them inside a HTML table. The value attribute

```
 value="#{cartBean.cartItemList}"
```

represents the list data over which h:dataTable iterates. The name of each individual item is specified through the var attribute. You can control the presentation style of each individual column through the columnClasses attribute.

Inside h:dataTable, h:inputText is used to take user input – the quantity of the current item:

```
<h:inputText value="#{cartItem.quantity}" size="5"/>
```

The attribute value="#{cartItem.quantity}" tells the JSF implementation to link the text field with the quantity property of the cart item. When the page is displayed, the getQuantity method is called to obtain the current property value. When the page is submitted, the setQuantity method is invoked to set the value that the user enters.

h:commandButton is used to create the "Update Cart" button, which allows the user to update the shopping cart. The

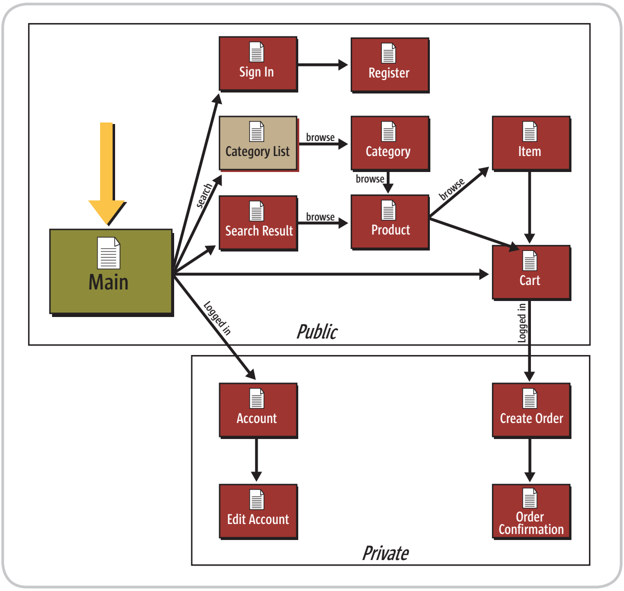


**Figure 1** Page-flow diagram

action attribute action="#{cartBean.updateAction}" contains a method-binding expression and tells the JSF implementation to invoke the updateAction method on the CartBean inside the session scope. The updateAction returns a navigation result, which determines the next page to go to.

The Check Out link is implemented with h:outputLink, which generates an HTML anchor element. Once the user clicks the link, it takes the user to the createOrder page.

### *Navigation*

Navigation is one of the key features provided by JSF. For this application, all navigation rules are defined inside the faces-navigation.xml. There are two navigation rules related to the current use case. Here's the definition of the first rule:

```
<navigation-rule>
  <from-view-id>/cart.jsp</from-view-id>
  <navigation-case>
    <from-outcome>success</from-outcome>
    <to-view-id>/cart.jsp</to-view-id>
  </navigation-case>
</navigation-rule>
```

This rule tells the JSF implementation that from the cart.jsp, if any action finishes successfully and returns success, the cart.jsp will be refreshed to reflect the current state of the shopping cart. The second rule is about error handling and is defined as a global navigation rule. It's not discussed here. Please refer to the faces-navigation.xml for detailed information.

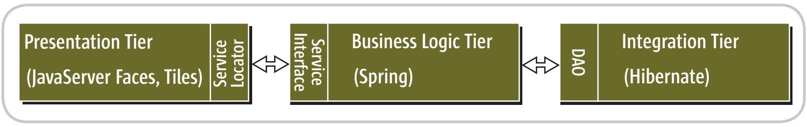


**Figure 2** High-level architecture diagram

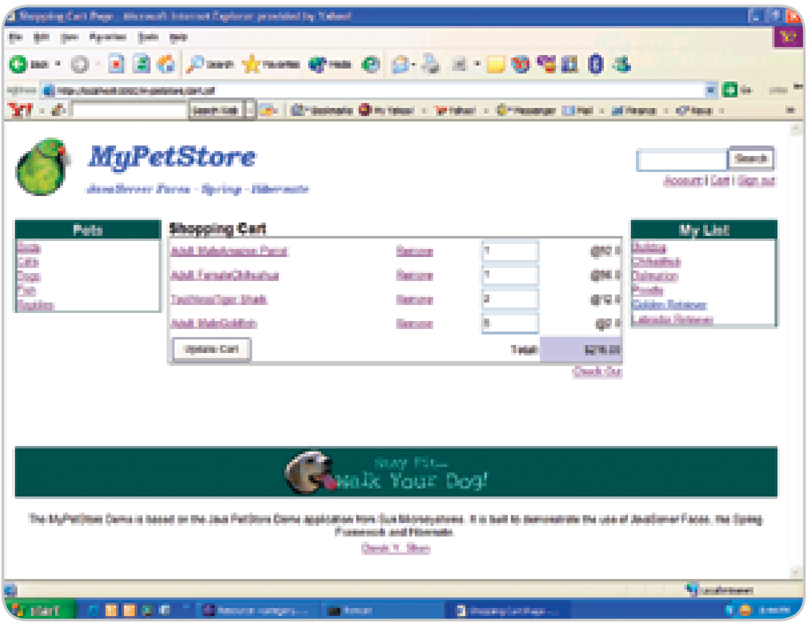


**Figure 3** Shopping cart screen

## Integration with Tiles

Tiles is a framework that makes using template layouts much easier through the use of a simple but effective tag library. Tiles separates the layout from content, makes your Web application easier to maintain, and keeps a common look and feel among all the pages. JSF and Tiles are a powerful combination.

JSF does not have built-in Tiles support. Tiles definitions cannot be referenced directly inside JSF applications. MyPetStore uses a workaround to integrate Tiles with JSF successfully. Tiles definition is referenced by JSF indirectly through a separate wrapper JSP file. The drawback of this approach is to have two JSP pages for each logical view. One is the content tile and the other is the wrapper JSP page with the Tiles definition. A tile is used to describe a page region managed by the Tiles framework. A tile can contain other tiles.

Here are step-by-step instructions on how to integrate JSF with Tiles:

- *Put struts.jar under your application's classpath* (Tiles is bundled with Struts1.2).
- *Enable Tiles inside web.xml* (see Listing 4). (Listings 4–10 can be downloaded from www.sys-con.com/java/sourcec.cfm.)
- *Build the layout template.* The layout of MyPetStore is defined inside layout.jsp (see Listing 5). This template controls the layout of the entire application. Tiles custom tag <tiles:insert attribute="sider" flush="false"/> tells the Tiles framework to insert the tile identified by the value of the specified attribute.
- *Write the base Tiles definition.* A Tiles definition allows you to specify the attributes that are used by a layout template. Tiles supports definition inheritance. You can declare a base definition and then create other definitions derived from that base. In MyPetStore, a base Tiles definition is defined inside the tiles.xml (see Listing 6). The base Tiles definition uses layout.jsp as the layout template and defines the common tiles used throughout the application, e.g., header, footer, etc.
- *Write the wrapper JSP page.* All the tiles with the real content are defined inside the Tiles directory. For each logical view, there's a wrapper JSP page. Here's the wrapper JSP page for cart.jsp:

```
<%@ taglib uri="http://jakarta.apache.org/struts/tags-tiles"
prefix="tiles" %>
<tiles:insert definition=".mainLayout">
  <tiles:put name="title" value="Shopping Cart Page"/>
  <tiles:put name="body" value="/tiles/cart.jsp"/>
</tiles:insert>
```

It tells the Tiles framework to insert the content tile /tiles/cart.jsp into the body part of the page defined by the base Tiles definition .mainLayout.

## Integration with the Business Logic Tier

The ServiceLocator pattern and the JSF bean management facility are used to integrate the JSF-based presentation tier with the business logic tier. The ServiceLocator abstracts the logic that looks for services. In this application, ServiceLocator is defined as an interface and implemented as a JSF managed bean, ServiceLocatorBean. The ServiceLocatorBean looks up

NEW VERSION

InstallShield®

✓ *Linux*

✓ *Unix*

✓ *Mac OS X*

✓ *Windows*

We do it all. Introducing InstallShield 10.5, the industry's choice for the strongest, most reliable installations that can run everywhere.

**Download Now!**

**www.installshield.com/JDJ**

**Special Offer!**

Receive expert training with purchase.
See Web site for details.

macrovision®

InstallShield, a Macrovision Company

the services from the Spring application context:

```
ServletContext context = FacesUtils.getServletContext();
this.appContext =
  WebApplicationContextUtils.getRequiredWebApplicationContext(con
text);
this.catalogService =
  (CatalogService)this.lookupService(CATALOG_SERVICE_BEAN_NAME);
…
```

The ServiceLocator is defined as a property inside the BaseBean. The JSF bean management facility wires the ServiceLocator implementation with those managed beans that need to access the middle tier.

```
<managed-property>
  <property-name>serviceLocator</property-name>
  <value>#{serviceLocatorBean}</value>
</managed-property>
```

IOC is used here.

## Middle Tier

The tasks in this tier consist of defining the business objects with their mappings, creating the service interfaces with their implementations, implementing the DAOs (data access object), and wiring the objects.

### *The Business Object*

The Cart business object is implemented as a POJO (plain old Java object) (see Listing 7). It contains the data associated with a shopping cart; it also contains actions with business logic, e.g., addItem, getSubTotal, etc.

### *The Business Service*

The CatalogService interface defines all of the catalog management–related services:

```
public interface CatalogService {
  public List getCategoryList() throws MyPetStoreException;
  public Category getCategory(String categoryId)
    throws MyPetStoreException;

  …
}
```

## Spring Configuration

Listing 8 provides the CatalogService's Spring configuration inside applicationContext.xml.

The Spring declarative transaction management is set up for the CatalogService. Spring bean factory creates and manages a CatalogService singleton object. By using Spring bean factory, the need for customized bean factories is eliminated.

The CatalogServiceImpl is the implementation of the CatalogService, which contains a setter for the CatalogDao. The CatalogDao is defined as an interface and its default implementation is the CatalogDaoHibernateImpl. Spring wires the CatalogServiceImpl with the CatalogDao. Because we are coding to interfaces, we don't tightly couple the implementations. For example, by changing the Spring applicationContext.xml, we can tie the CatalogServiceImpl with a different CatalogDao implementation – CatalogDaoJDOImpl.

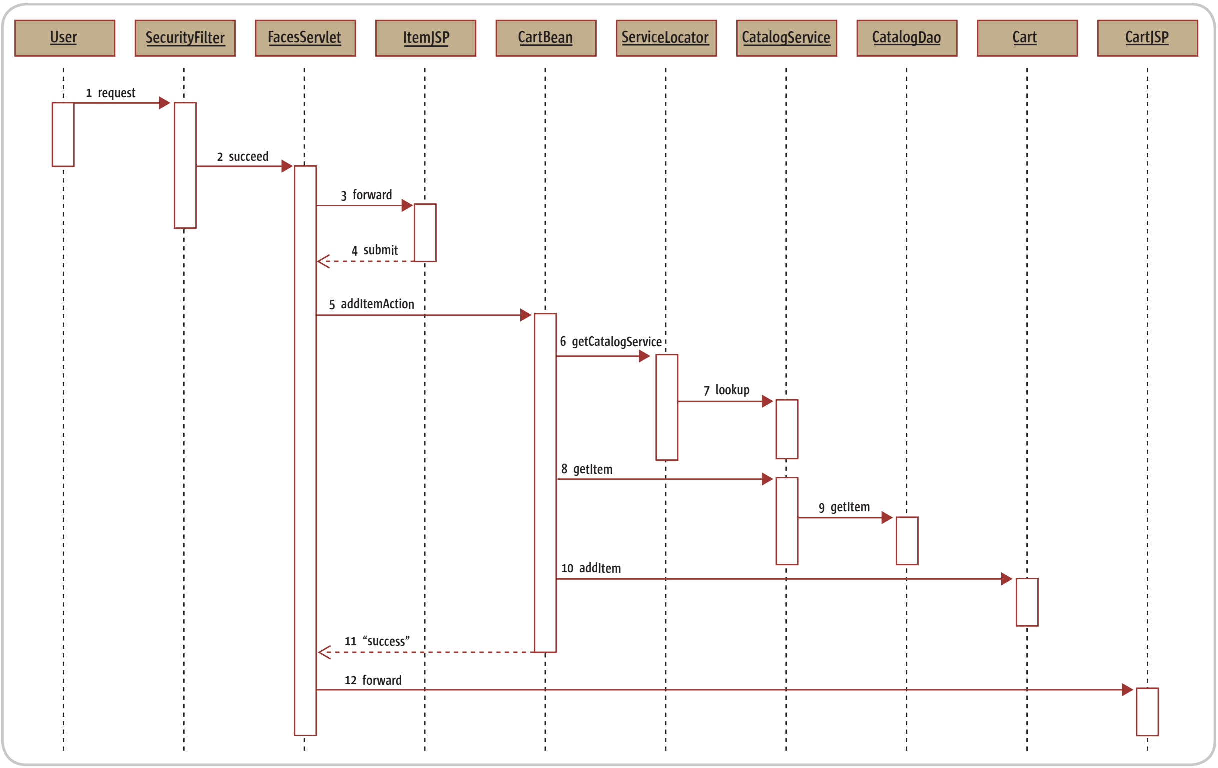
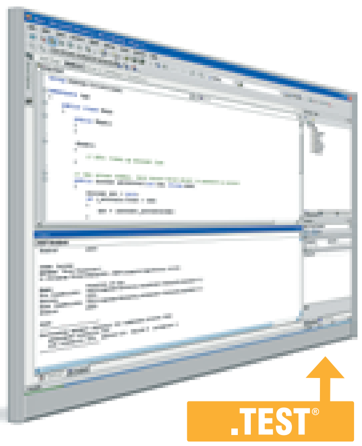


**Figure 4** A sequence diagram of the AddItemToCart use case

# Achieve higher quality

## in less time, with fewer resources.



## Parasoft® Automated Error Prevention (AEP) Products:

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

**Parasoft AEP Products ensure that comprehensive error prevention practices are applied consistently and uniformly across your entire team.**

By automating compliance to a unified set of testing and coding standards, Parasoft AEP Products enable every team member to follow the same best practices and the same error prevention techniques – including coding standards analysis, unit testing, code review and regression testing.

**Automated testing and standards compliance for any team configuration.**

Parasoft AEP Products automate error prevention for Java, C/C++, Web Services, the Microsoft® .NET Framework, Embedded Development, Web Development, Database Development and more.

**For details go to: www.Parasoft.com/AchieveQuality**

**Call: 888-305-0041 x3307 Email: AchieveQuality@Parasoft.com**

PARASOFT®
We make software work.™

*Founded in 1987, Parasoft's clients include IBM, HP, DaimlerChrysler and over 10,000 companies worldwide.*

**Availability**

Parasoft AEP Products are available on Linux, Solaris, and Windows NT/2000/XP.

**Contact Parasoft for details and pricing information.**

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

## Integration with Hibernate

Listing 9 provides the HibernateSessionFactory's configuration. CatalogDao uses the HibernateTemplate to integrate Spring with Hibernate. Here's the configuration for HibernateTemplate:

```
<bean id="hibernateTemplate"
  class="org.springframework.orm.hibernate.HibernateTemplate">
  <property name="sessionFactory">
    <ref bean="sessionFactory"/>
      </property> <property name="jdbcExceptionTranslator">
    <ref bean="jdbcExceptionTranslator"/>
  </property>
</bean>
```

Hibernate maps business objects to the relational database using XML configuration files. Item.hbm.xml expresses the mapping for the Item business object. The configuration files are in the same directory as the corresponding business objects. Listing 10 provides the Item.hbm.xml. Hibernate maintains the relationship between objects. The mapping definition defines a many-to-one relationship between Item and Product.

Finally, CatalogDao is wired with HibernateTemplate by Spring:

```
<bean id="catalogDao"
  class="mypetstore.model.dao.hibernate.CatalogDaoHibernateImpl">
  <property name="hibernateTemplate">
    <ref bean="hibernateTemplate"/>
  </property>
</bean>
```

### *End-to-End*

Figure 4 demonstrates the end-to-end integration of all the tiers for AddItemToCart: a sub-use case of the UpdateShoppingCart use case.

### *Technologies and JSF Features Matrix*

The UpdateShoppingCart use case doesn't cover all the technologies and advanced JSF features we used in this application, e.g., security, pagination, etc. Table 1 can serve as a reference catalog to help you understand this application.

| | | |
|---|---|---|
| **Business Logic Tier Integration** | | |
| | *Helper Classes* | mypetstore.view.servicelocator.ServiceLocator |
| | *Configuration* | faces-managed-beans.xml, applicationContext.xml |
| | *Managed beans* | mypetstore.view.bean.ServiceLocatorBean<br>mypetstore.view.bean.BaseBean |
| | *Description* | *ServiceLocator* pattern and JSF bean management facility are used. |
| **Chaining bean definitions** | | |
| | *Configuration* | faces-managed-beans.xml |
| | *Managed beans* | mypetstore.view.bean.ServiceLocatorBean<br>mypetstore.view.bean.BaseBean |
| | *Description* | JSF implementation is an IOC container.<br>The implementation of the ServiceLocator is chained with other managed bean definitions. |
| **Custom Validator** | | |
| | *Helper classes* | java.util.regex.Pattern<br>mypetstore.view.validator.regex.RegexValidator<br>mypetstore.view.validator.regex.RegexValidatoTag |
| | *Configuration* | mypetstore.tld, faces-config.xml |
| | *JSP pages* | createAccount.jsp |
| | *Description* | Using the regular expression validator to demonstrate how to build a validator with custom tag. |
| **Error Message Customization** | | |
| | *Configuration* | mypetstore.view.bundle.Messages.properties<br>faces-config.xml |
| | *Description* | Message bundle is defined to customize the JSF build-in error messages |
| **h:dataTable** | | |
| | *JSP pages* | category.jsp, product.jsp, mylist.jsp.searchResult.jsp |
| **Initializing managed bean property from request parameters** | | |
| | *Configuration* | faces-managed-beans.xml |
| | *Managed beans* | mypetstore.view.bean.CategoryPageBean<br>mypetstore.view.bean.ProductPageBean<br>mypetstore.view.bean.ItemPageBean<br>mypetstore.view.bean.SearchPageBean |
| | *Description* | JSF implementation is able to retrieve request parameter and set it as property inside managed bean. |
| **Pagination** | | |
| | *Helper classes* | org.springframework.beans.support.PagedListHolder |
| | *Managed beans* | mypetstore.view.bean.BasePaginationBean |
| | *JSP pages* | category.jsp, product.jsp, mylist.jsp, searchResult.jsp |
| | *Description* | Spring PagedListHolder is used to build the pagination.<br>The pagination logic is inside BasePaginationBean. |
| **Security** | | |
| | *Helper classes* | mypetstore.view.util.SecurityFilter |
| | *Configuration* | web.xml |
| | *JSP pages* | signonRedirect.jsp |
| | *Description* | Standard Servlet Filter is used to implement the authentication.<br>signonRedirect.jsp is used to redirect the user to his original target URL after sign on. |
| **Tiles Integration** | | |
| | *Configuration* | web.xml, tiles.xml |
| | *JSP pages* | layout.jsp, all JSP wrapper pages |
| | *Description* | struts.jar needs to be under the classpath. |

**Table 1** Technologies and JSF features used in MyPetStore application

### *Configuration and Installation*

You can download the source code for MyPetStore from www.sys-con.com/java/sourcec.cfm.

### *The Package Structure*

After unzipping the code, you should see the following directory structure:

```
mypetstore
  /bin
  /docs
  /lib
  /src
  /web
    /images
    /tiles
    /WEB-INF
  build.xml
```

Table 2 explains each part of the directory structure.

### *System Requirements*

You'll need the following tools to build and run the MyPetStore application:

- JDK 1.4.2 or later
- Ant
- Tomcat 5.x
- MySQL 4.x

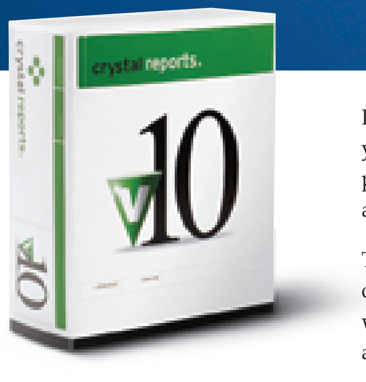
# We'd like to think that not all perfect matches are made in heaven.

## Crystal Reports 10

Which edition of Crystal Reports® is right for you?

| | Report Author/IT Editions | | Bundled Developer Editions | | Full Developer Editions | |
|---|---|---|---|---|---|---|
| | Standard | Professional | .NET Edition [2] | Java® Edition [3] | Developer | Advanced |
| **Report Creation** | | | | | | |
| Visual report designer for rapid data access and formatting | ● | ● | ● [1] | ● [1] | ● | ● |
| Customizable templates for faster, more consistent formatting | ● | ● | | | ● | ● |
| Repository for reuse of common report objects across multiple reports[4] | | ● | | | ● | ● |
| **Data Access** | | | | | | |
| PC-based and Microsoft® ODBC/OLE DB for MS Access and SQL Server | ● | ● | ● | ● | ● | ● |
| Enterprise database servers (ODBC, native) | | ● | ● [1] | ● [1] | ● | ● |
| Custom, user-defined data through JavaBeans™ | | | | ● | ● | ● |
| Custom, user-defined data through ADO and .NET | | | ● | | ● | ● |
| **Report Integration** | | | | | | |
| Report viewing APIs (.NET and COM SDKs) | | | ● | | ● | ● |
| Report viewing APIs (Java SDK) | | | | ● | ● | ● |
| Extensive report viewer options (DHTML, ActiveX, Java Plug-in, and more) | | | | | ● | ● |
| APIs for run-time report creation and modification | | | | | | ● |
| Report Parts for embedding report objects in wireless and portal apps | ● | ● | | | ● | ● |
| **Report Deployment** | | | | | | |
| Crystal Reports components for report viewing, printing, and exporting: | | | | | | |
| a) Java reporting component | | | | ● | ● | ● |
| b) .NET reporting component | | | ● | | ● | ● |
| c) COM reporting component | | | | | ● | ● |
| Full featured report exporting | | ● | | | ● | ● |
| Report server (Crystal Enterprise Embedded deployment license) | | | | | | ● |

1 Limited functionality. 2 Bundled with Microsoft® Visual Studio® .NET and Boland® C#Builder™.
3 Bundled with BEA WebLogic Workshop™ and Boland® JBuilder®. 4 This feature is available on the Crystal Enterprise CD, included in the Crystal Reports 10 package.



Perfect matches can be made here too. In order to quickly determine which Crystal Reports® best suits your project requirements, we've provided this basic feature chart. Crystal Reports® 10 simplifies the process of accessing, formatting, and tightly integrating data into Windows and web applications via an enhanced designer, flexible data connectivity options, and rich Java™, .NET, and COM SDKs.

To learn more about Crystal Reports 10, compare over 150 different features across versions, or to access technical resources like the Developer Zone and evaluation downloads, visit: www.businessobjects.com/dev/p7. To ask more specific report project related questions, contact an account manager directly at 1-888-333-6007.

BUSINESS OBJECTS®

| Folder or File | Content |
|---|---|
| bin | database script |
| dist(after build) | war file |
| docs | installation instruction and Java docs |
| lib | library jar files |
| src | Java source, resource bundle and Hibernate mapping files |
| web | web application files |
| images | images |
| tiles | JSP templates |
| WEB-INF | JSF, Spring, Tiles, Custom Tag, and web application configuration files |
| build.xml | ant build script |

**Table 2** MyPetStore application package content

*Installation Instructions*

- *Prepare the database.* Run bin/mypetstore.sql against MySQL to build database schema and load seed data. If you'd like to use another RDBMS, this script may need to be modified.
- *Change the configuration* (optional for default MySQL installation).
  - Find and open web/WEB-INF/applicationContext.xml.
  - Under the definition of the datasource bean, modify the JDBC connection properties (URL, username, pass word) to match the database you're using.
  - Save the file.
- *Build the Web application by Ant* (the default target is build. war).
- *Copy the dist/mypetstore.war to the <tomcat-home>/webapps directory.*
- *Start Tomcat.*

For the default Tomcat installation, MyPetStore is accessible from http://localhost:8080/mypetstore

Log on to the application using j2ee as the username and password.

## Summary

In this article I tried to provide a picture of how you can integrate JSF, Tiles, Spring, and Hibernate in one application. With minimum theory, you should now be able to start building such typical Web applications as a pet store. If I sparked your interest in studying any of these technologies, my mission was accomplished.

## Resources

- *JavaServer Faces:* http://java.sun.com/j2ee/javaserverfaces
- *Tiles (bundled with Struts1.2):* http://struts.apache.org
- *The Spring Framework:* http://springframework.org
- *Hibernate:* http://hibernate.org
- *MySQL:* http://mysql.com/
- *Tomcat:* http://jakarta.apache.org/tomcat/
- *Java Pet Store:* http://java.sun.com/developer/releases/petstore
- Johnson, R. (2002). *Expert One-on-One J2EE Design and Development (Programmer to Programmer).* Wrox.
- Singh, I., et al, (2002). *Designing Enterprise Applications with the J2EE Platform.* Addison-Wesley Professional.

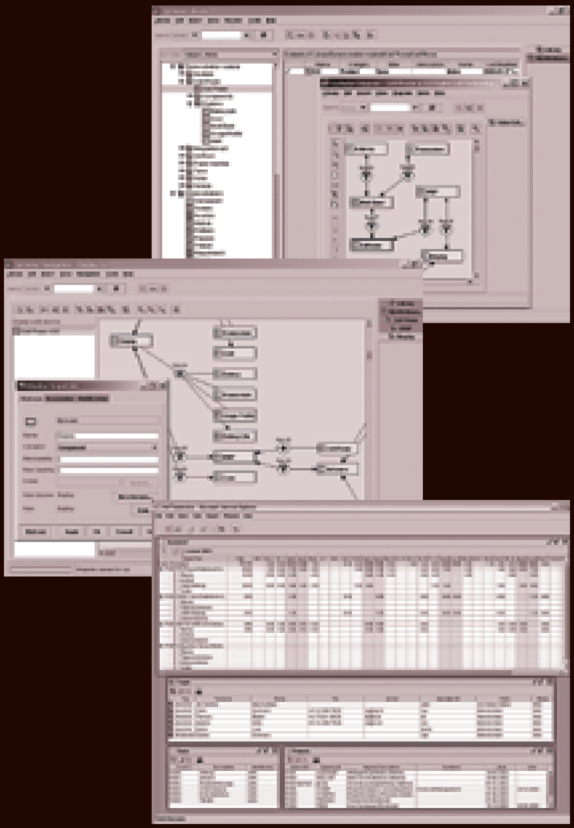
**Listing 1**
```
public class CartBean extends BaseBean {
  private Cart cart;

  …
  public String addItemAction() {
    String itemId =
      FacesUtils.getRequestParameter(ITEM_ID_PARAMETER_NAME);
    try {
      Item item =
        this.getServiceLocator().getCatalogService().
           getItem(itemId);
      this.cart.addItem(item);
    } catch(MyPetStoreException me) {
      String msg = "Could not add item to cart ";
      FacesUtils.addErrorMessage(msg + ": Internal Error");
      return NavigationResults.FAILURE;
    }
    return NavigationResults.SUCCESS;
  }

  …
  public Cart getCart() {
    return this.cart;
  }
}
```

**Listing 2**
```
<managed-bean>
  <description>Backing bean for the shopping cart.</description>
  <managed-bean-name>cartBean</managed-bean-name>
  <managed-bean-class>
    mypetstore.view.bean.CartBean
  </managed-bean-class>
  <managed-bean-scope>session</managed-bean-scope>
  <managed-property>
    <property-name>serviceLocator</property-name>
    <value>#{serviceLocatorBean}</value>
  </managed-property>
</managed-bean>
```

**Listing 3**
```
<%@ taglib prefix="f" uri="http://java.sun.com/jsf/core" %>
<%@ taglib prefix="h" uri="http://java.sun.com/jsf/html" %>

<f:subview id="cart">
<h:form id="cartForm">
<h:outputText value="Your Shopping Cart is Empty"
  styleClass="title"      rendered="#{cartBean.numberOfItems <=
0}"/>
<h:outputText value="Shopping Cart" styleClass="title"
  rendered="#{cartBean.numberOfItems > 0}"/>
<h:panelGrid columns="1" styleClass="box"
  rendered="#{cartBean.numberOfItems > 0}">
<h:dataTable id="cartTable" value="#{cartBean.cartItemList}"
  var="cartItem" styleClass="standard"
  columnClasses="cartColumn1, cartColumn2, cartColumn3, cartCol-
umn4">
  <h:column>
    <h:outputLink  value="item.jsf?itemId=#{cartItem.item.
itemId}">
      <h:outputText value="#{cartItem.item.attribute1}" />
      <h:outputText value="#{cartItem.item.productName}"/>
    </h:outputLink>
  </h:column>

  …
  <h:column>
    <h:inputText value="#{cartItem.quantity}" size="5"/>
  </h:column>

  …
</h:dataTable>

…
  <h:commandButton value="Update Cart"
    action="#{cartBean.updateAction}"/>

…
</h:panelGrid>
<h:panelGrid styleClass="standard" columnClasses="rightAlign"
rendered="#{cartBean.numberOfItems > 0}">
  <h:outputLink  value="createOrder.jsf">
    <h:outputText value="Check Out" />
  </h:outputLink>
</h:panelGrid>
</h:form>
</f:subview>
```

# Rich Thin Clients for J2EE

## UltraLightClient offers a server-side API to Swing, providing rich GUIs for J2EE applications.



- **Server-side programming model:** develop scalable web applications for thousands of users as simply as stand-alone Swing applications.
- **Superior security:** no application code is executed on the client, nothing is stored in a browser cache.
- **Application deployment on server:** a lean Java presentation engine on the client serves any number of applications.
- **Pure Java library:** use your favorite IDE and get add-on tools for visual editing, client/server simulation, and load/performance testing.

Ready for IBM WebSphere Studio software

canoo.com

Canoo Engineering AG http://www.canoo.com/ulc/

**Download your free trial today!**

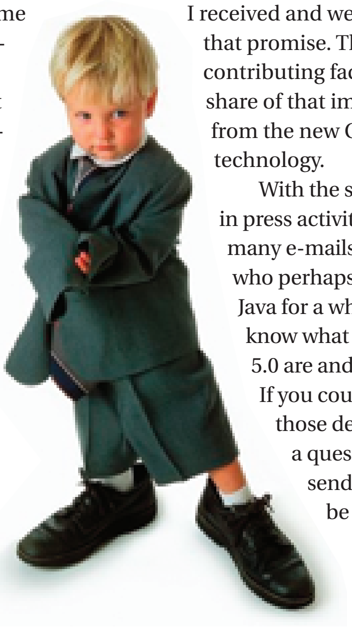
**Calvin Austin**
Core and Internals Editor

# J2SE 5.0 Ready for Business

I am pleased to announce that the J2SE 5.0 release has gone final and is ready for you to download! The first set of downloads for Windows, Solaris, and Linux are available from the http://java.sun.com/j2se/5.0 Web site. This even includes a 64-bit AMD64 port on Linux for server-side applications. Other OS support and tools will follow from our partners, so please let them know that you are waiting for a particular port.

I was fortunate to be able to fly to New York for the J2SE 5.0 launch and give a short presentation, called "5 reasons to move to 5," to the New York SIG. I checked the flight deals on Orbitz (which runs J2SE on Linux), sent my expense request using a Java Web Start-enabled expense tool, and I was ready to go.

While I was waiting for the plane to take off, I paged through the in-flight magazine and came across the section describing the type of aircraft I was currently seated on. It proudly noted that the engine for this 757 flight was a Rolls Royce. I've never actually been in a Rolls Royce car, which is now part of the BMW group; however, the name Rolls Royce for me is a byline for reliability and performance. I think their messaging worked; I knew they wouldn't really put any old engine on a plane but I felt more assured that a Rolls Royce engine would have exceeded any certification test they threw at it and was more than capable of getting me to my destination.



Fast forward to the New York SIG. As Yakov writes in his editorial this month, Java is used heavily on Wall Street. I knew Java was behind many of the popular consumer Web sites but the Java platform is the Rolls Royce engine for Wall Street and provides not just the horsepower, but the integration solution too. The good news is that the J2SE 5.0 release is the most rock-solid, stable platform we have ever tested.

The testing criteria is raised for each release and our testing experts and TCK specification test writers have done an impressive job. At the same time we have added more features, made it faster to start up and also smaller to download. That may be a pleasant surprise for many of you – improving start up time was the number one request I received and we made good on that promise. There were several contributing factors but the lion's share of that improvement came from the new Class Data Sharing technology.

With the subsequent volume in press activity, I've received many e-mails from developers who perhaps haven't looked at Java for a while and want to know what the advantages of 5.0 are and should they switch? If you count yourself as one of those developers and have a question or comment, send me an e-mail, I will be happy to help.

To continue with the Tiger theme, we have two great J2SE 5.0 articles this month. The first is a comprehensive look at the new enumerated type keyword, developed through JSR 201. The second is an article describing the monitoring and management framework as developed through JSRs 163 and 174. This is just the start of some great J2SE 5.0 content we have lined up, so make sure you pick up next month's magazine too!

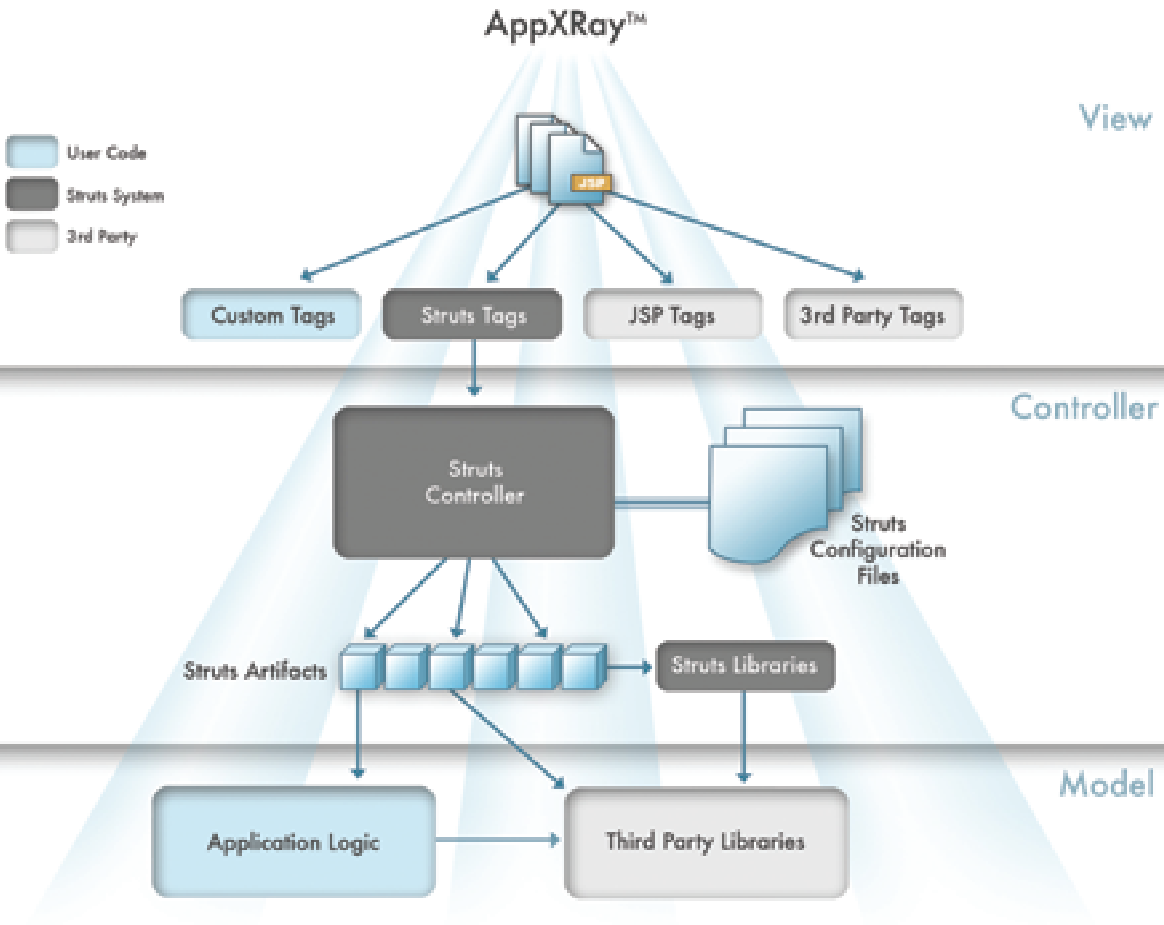
To close, I just wanted to share another new 5.0 feature with you. The feature in question is the new StringBuilder class, also known as the unsynchronized StringBuffer. At first glance you may feel that synchronization generally means slower and therefore you should retrofit this new API in your codebase to see dramatic speed improvements. Hold on though; if you have only a single thread accessing your StringBuffer in the first place (the conditions that StringBuilder requires), the monitor lock itself is fairly cheap. Monitors only start to get expensive when more than one thread needs access to that protected code block. The first thread just tests and sets a bit and proceeds. It's only if another thread passes through that same piece of code at the same time that things change. The next thread detects the bit is set and creates a monitor queue to hold any other threads that need to wait. This is the more expensive operation and is called monitor inflation. In my own tests with a single thread that never required monitor inflation I needed to run at least 10,000 StringBuffer operations to see a savings. As in all performance tweaks, I would recommend running your own benchmarks to see what savings you can achieve.

"The Java platform is the Rolls Royce engine for Wall Street and provides not just the horsepower, but the integration solution too"

A co-editor of *JDJ* since June 2004, **Calvin Austin** is the J2SE 5.0 Specification Lead at Sun Microsystems. He has been with Java Software since 1996 and is the Specification Lead for JSR-176, which defines the J2SE 5.0 ("Tiger") release contents.

*calvin.austin@sys-con.com*

# Do you understand all the relationships in your web application?



**Nitrox for JSP** | **Nitrox for Struts** | **Nitrox for JSF**

## NitroX™ AppXRay™ Does!

- **JSP debugging including stepping in/out of JSP tags and unique JSP variables view**
- **Source and Visual JSP & Struts editors with knowledge of all web app layers**
- **Design time validation, consistency and error checking for JSP source,Tag Libs, Struts and Server-based Java code in a web app**
- **Eclipse & IBM WSAD, WSSD plug-in**

**Download a free, fully functional trial copy at: www.m7.com/d7.do**

# Distributed Notification with Java RMI

*Server-to-client communication*

by Michael J. Remijan

Java's implementation of Remote Method Invocation (RMI) is easy to use and powerful. Java makes setting up an RMI server an almost trivial task because the JVM handles complex tasks such as networking and object serialization. Once running, connecting client applications to the RMI server is also a breeze.

There are numerous examples and how-to articles for client-to-server communication (http://java.sun.com/developer/onlineTraining/rmi), but what about the other way? Is it possible for an RMI server to actively communicate with all the clients that are connected to it without the client initiating the conversation? In other words, is distributed notification possible? The purpose of this article is to demonstrate that yes, it is possible. (The source code for this article can be downloaded from www.sys-con.com/jdj/sourcec.cfm.)

RMI applications are driven by the need to have centrally located business logic used by multiple clients at the same time. This presents a number of problems, such as clients not knowing what other clients are doing. A simple example that immediately demonstrates this problem is caching data on the client side for fast reuse. When one client updates information, all the other clients are now working with old data.

The typical – and not the best – solution for this problem is to have each client poll the RMI server for updates on a regular basis. First, constant polling puts unnecessary strain on the RMI server because it's forced to handle additional network traffic. Second, there will be a period of time when the client is working with old data. In the J2EE world, Java Messaging Service (JMS) solves this problem the best. Assuming all data updates are done at a central location, a JMS message can be easily sent to all registered clients when an update occurs. When a client receives the message, the cache can be refreshed.

**Michael J. Remijan** is a research programmer at the National Center for Supercomputing Applications (NCSA). His chief responsibilities include data mining and access to terabyte-sized astronomy data sets. Michael received a BS in mathematics and computer science from the University of Illinois and is currently working on an MBA in technology management.

*mjremijan@yahoo.com*

An alternative solution is to keep plain RMI and follow the event notification model similar to AWT/Swing. In those models, an object implements a simple listener interface and is then added to an appropriate event notification list. When that event occurs, every object in the list becomes notified of the event. The object takes whatever action necessary in response to the event. Such a model obviously is a better solution than polling, and solves the problems that polling has – there is no unnecessary network traffic and clients are notified instantly when a change in the data has occurred.

Applying this model to RMI is not trivial. Consider an RMI-based time server as an example, where client applications register with an RMI server. Every second the RMI server fires an event to inform the clients of the new time. To successfully do this, first define a remote interface named TimeServices.

```
import java.rmi.*;
public interface TimeServices extends
Remote
{
    public void addTimeMonitor(TimeMonitor
      tm)
    throws RemoteException;
}
```

The TimeServices interface declares the methods used for client-to-server communication (see Figure 1).

The Remote interface has one method, addTimeMonitor (TimeMonitor), which is used by RMI client applications to register themselves with the RMI server. This is analogous to the setActionListener() methods in Swing only instead of implementing the ActionListener interface, an object that implements the TimeMonitor interface is needed. Just as the ActionListener interface serves to link an event with the application code that processes the event, the TimeMonitor Interface serves as a way for the RMI Server to talk back to its clients.

```
import java.rmi.*;
import java.util.Date;
public interface TimeMonitor extends Remote
{
    public void setTime(Date d)
    throws RemoteException;
}
```

Because of this, TimeMonitor is a remote interface that allows server-to-client communication (see Figure 2).

Implementing the TimeMonitor interface becomes a challenge, because it's not possible for every client to create its own implementation and use that implementation to register with the RMI server. When the client tries to register with the server by calling the addTimeMonitor() method, the TimeMonitor parameter serializes to a byte stream and is transmitted over the network to the server. The server then needs to know how to deserialize the stream and reconstitute the object. Even though syntactically the TimeMonitor interface is needed to compile, when the server is actually running it needs the implementation of the TimeMonitor interface in the CLASSPATH in order to deserialize the TimeMonitor objects correctly. If the implementation was not in the CLASSPATH, a ClassNotFoundException is thrown when the RMI server attempts to deserialize the stream.

To get around this problem, the client must use an implementation of TimeMonitor that the RMI server provides.

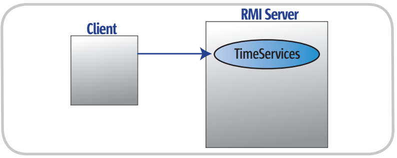


**Figure 1** Client-to-server communication

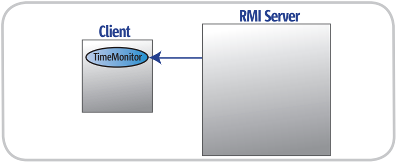
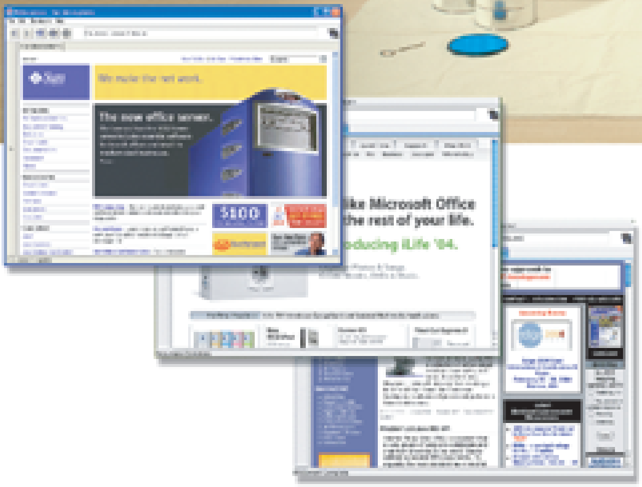


**Figure 2** Server-to-client communication

# WebRenderer™



## Standards compliant embeddable web browser component

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

**Why WebRenderer?**

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

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

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

JadeLiquid™ Software

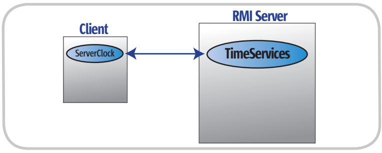


**Figure 3** Two-way communication between client and server

Assume this implementation is a class named ServerClock (see Listing 1). Since ServerClock implements the TimeMonitor interface, it's easy to have new instances of this object register itself with the RMI server. The constructor of the ServerClock class can do just this. Once registered, communication can go back and forth between the client and server (see Figure 3).

The client would create an instance of this class by supplying the name and port number of the RMI server.

```
ServerClock serverClock = new
ServerClock(serverName, port);
```

The ServerClock constructor performs two critical operations. First, it makes this method call:

```
UnicastRemoteObject.exportObject( this );
```

According to the J2SE documentation, this method call dynamically "exports the remote object to make it available to receive incoming calls using an anonymous port." Making this method call is the key that allows the server to communicate with all the various clients without requiring stub classes from the client, or knowing the names and ports of client machines.

Second, the constructor contacts the RMI server and registers the object by calling the addTimeMonitor() method:

```
timeServices.addTimeMonitor( this );
```

This doesn't result in a ClassNotFoundException because the keyword "this" refers to an instance of the ServerClock object that the RMI server provided.

Now the ServerClock class needs to implement the setTime(Date) method of the TimeMonitor class:

```
/**
 * TimeMonitor interface method
 */
public void setTime(Date d)
{
    System.out.println("The new time
      is: " + d);
}
```

This implementation of setTime (Date d) will technically work, but the RMI client will never know that the RMI server called this method because all the method does is print the Date object to standard out. The RMI client needs to be notified by the ServerClock whenever the setTime(Date d) method is called. A simple solution is for the RMI server to define another, nonremote interface SecondsListener:

```
import java.util.Date;

public interface SecondsListener
{
    public void tick(Date d);
}
```

then add a method to ServerClock that stores implementations of SecondsListener in a vector.

```
/**
 * add listener to list
 */
public void addSecondsListener(Seconds
  Listener sl) {
    synchronized(listeners) {
        if (!listeners.contains(sl)) {
            listeners.add(sl);
        }
        listeners.notifyAll();
    }
}
```

Finally, ServerClock's implementation of the setTime(Date d) method is changed to loop over the vector.

```
/**
   * TimeMonitor interface method
   */
  public void setTime(Date d)
  {
      synchronized(listeners) {
          for (
              Iterator itr=listeners.
                iterator();
              itr.hasNext();
              ((SecondsListener)itr.
                next()).tick(d)
          ) {}
          listeners.notifyAll();
      }
  }
```

With these additions in place, the client can create as many listener classes as are needed and register the classes with ServerClock:

```
ServerClock serverClock = new
ServerClock(serverName, port);
serverClock.addSecondsListener(new
ListenerA());
serverClock.addSecondsListener(new
ListenerB());
serverClock.addSecondsListener(new
ListenerC());
…
```

Since the SecondsListener implementations exist on the client, there is no problem with ClassCastExceptions.

## Summary

This article successfully shows the event notification model applied to RMI so an RMI server can notify registered clients of events as easily as a JButton notifies its ActionListeners when the button is pushed.

The RMI server is the place where events are generated and fired. A remote interface (TimeServices) allows an RMI server–supplied object (ServerClock) to add itself to an event notification list on the server. Another remote interface (TimeMonitor) allows the server to communicate back with the client when the event occurs. The server-supplied object (ServerClock) allows the client to add an arbitrary number of local objects implementing a nonremote, server-supplied interface (SecondsListener). When the RMI server fires an event (TimeMonitor.setTime(Date)), the server-supplied object (ServerClock) loops over its list of registered listeners and passes the event along.

**Listing 1**

```
public class ServerClock implements TimeMonitor
{
     private Vector listeners;
     private String hostName;
     private int port;
     /**
      * constructor
      */
     public ServerClock(String hostNameVal, int portVal)
     throws Exception
     {
          hostName = hostNameVal;
          port = portVal;
          UnicastRemoteObject.exportObject( this );

          String  serverName = "rmi://" + hostName + ":" +
port + "/TimeServices" ;
          TimeServices timeServices = null;
          try {
               timeServices = (TimeServices)Naming.
lookup(serverName);
          }
          catch ( Exception e ) {
               e.printStackTrace();
               throw e;
          }
          timeServices.addTimeMonitor( this );
     }
```

Enterprise-Class Open Source™ Has Arrived

***Integrated, Certified & Supported***

**Java™ Application Platform**

GLUECODE SOFTWARE

Based upon trusted software components from the Apache Software Foundation™, Gluecode JOE™ consolidates your application needs into a single Java-certified platform.

Download Gluecode JOE today at www.gluecode.com.

# BUILDING MANAGEABILITY

*Using the management and monitoring APIs to build application manageability*

by Satadip Dutta

*Your application has finally been tested and deployed in a production environment. However, the IT operators are complaining that the application is consuming more system resources than originally expected. The problem being described by the IT operators cannot be repeated in the development environment. To establish the cause of these problems, it's important to get enough information about the runtime environment, more specifically what is going on inside the Java Virtual Machine (JVM).*

Before J2SE 5.0, the information available from the JVM was at best extremely limited. JSR 174 has added management and monitoring APIs in J2SE 5.0 that expose valuable JVM information. The exposed information ranges from JVM health indicators like memory and threads to class loading and garbage collection information. This article provides an introduction to manageability and describes how the monitoring and management APIs can be used to externally manage Java applications as well as build manageability in applications.

**Satadip Dutta** is a software architect at Hewlett-Packard (http://devresource.hp.com) and has been programming in Java since 1997. He is a committer for the XMLBeans project (http://xml.apache.org/xmlbeans). His areas of interests include distributed software architecture, Web services, and user interface design. Satadip holds an MS in computer science from Virginia Tech.

*sdutta@vt.edu*

## Introduction to Manageability

Manageability is defined as the capability that allows an application to be managed and controlled. Manageable applications provide mechanisms by which it is possible to probe, measure, track, and control it (see Figure 1). Typically these actions are:

- ***Monitoring:*** To capture runtime information from the application
- ***Tracking:*** To observe aspects of an application over a period of time
- ***Control:*** To alter the behavior of a runtime component

A variety of technologies are available to manage applications, beginning from basic logs and SNMP to newer standards like Java Management Extensions (JMX) and Web Services Distributed Management (WSDM). Java Management Extensions allows applications to be managed by management software by providing an abstraction layer between the application and the management software. WSDM is a relatively new management protocol that uses Web services technology (WSDL/SOAP/UDDI) to manage distributed resources. WSDM is platform agnostic and can manage any resource (applications, Web services and their end points) that can be exposed as a Web service.

This article focuses on JMX as the technology to manage applications. The new monitoring and management APIs use JMX and reside in the java.lang.management package. The APIs expose data in areas like memory, thread, runtime, operating system, garbage collection, etc. JMX can be used to expose coarse business–level information to fine-grained information about specific classes and objects. The information exposed by JMX can be consumed via system management software like HP Openview, IBM Tivoli, or simpler tools like jconsole.

## Basics of JMX

The JMX specification defines an API and architecture that provides a standard mechanism to management-enable Java applications. The JMX architecture (see Figure 2) has three layers:

- Instrumentation
- Agent
- Distributed

### *Instrumentation Layer*

The instrumentation layer consists of management beans (MBeans) and managed resources that are instrumented with MBeans. This layer is responsible for encapsulating management resources with an interface similar to JavaBeans. Management resources are attributes of an application that will help in determining the state of the application. This includes system-level information (like the number of threads executing), business logic–related information (like the numbers of orders being processed in shopping carts), and service-level information (like uptime and response time). The JMX specification defines four types of MBeans: Standard, Dynamic, Open, and Model MBean.

*Agent Layer*

The agent layer defines a JMX agent, which is a management entity that runs on a JVM. The agent acts as the liaison between the MBeans and the management application. A JMX agent is composed of an MBeans server, a set of MBeans representing managed resources, a minimum number of agent services implemented as MBeans, and typically at least one protocol adaptor or connector.

The MBean server is the keystone of the JMX architecture and the central registry for MBeans. All management operations on MBeans are brokered through the MBean server. The MBean server, MBeans, and adapters make use of the generic functionality that is available as services. Additional services can be dynamically added by the application or the management system to extend the functionality of a JMX agent. Some of the services included are monitoring, timer, relation, and dynamic class loading.

*Distributed Layer*

The distributed layer typically contains at least one protocol adapter or connector. The adapters and connectors make the agent accessible remotely. Adapters provide a view of the JMX agent through a protocol like HTTP or SNMP. Connectors, on the other hand, are used to connect the JMX Agent with a remote JMX-compliant management application using a distributed technology like RMI. The distributed layer also has tools that expose the management view of a JMX agent and the MBeans via another protocol like SNMP, and tools that interact with the JMX agent and MBeans via a connector for distributed applications. This layer is responsible for providing connection services, security, and consolidated management information from disparate JMX agents. The adapter tools interact with the MBeanServer to generate artifacts that are required by a particular protocol. For example, MOF files may be created by a Web Based Enterprise Management (WBEM) adapter tool.

## Using the Monitoring and Management APIs

The monitoring and management APIs use JMX as the underlying mechanism to expose the JVM information. The use of JMX allows the information to be available locally and remotely to applications that support JMX. The performance impact of extracting the JVM information is extremely low. The information exposed by new APIs can be used to aid in:

- ***External monitoring and management:*** Allows external entities like management software, system administrators, IT operators, support engineers, and developers to monitor the JVM and Java applications.
- ***Internal monitoring and management:*** Allows the developers to add logic to self-monitor and manage the JVM to make the applications more manageable and robust.

## External Monitoring and Management

From the perspective of IT operators, the top two monitoring and tracking aspects for Java applications are memory consumption and CPU usage. Table 1 outlines the management interfaces that are available to monitor and track memory and CPU consumption.

The JMX agent available on the JVM is disabled by default. To enable out-of-the box local JMX monitoring, the application needs to be started with the following arguments.

```
$JAVA_HOME/bin/java -Dcom.sun.management.jmxremote
ApplicationName
```

This will start the JMX agent and allow the application to be monitored from the local machine using an application such as jconsole.

To enable out-of-the-box remote monitoring without SSL and password authentication the following command line can be used.

```
$JAVA_HOME/bin/java -Dcom.sun.management.jmxremote.
port=1097
-Dcom.sun.management.jmxremote.authenticate=false
-Dcom.sun.management.jmxremote.ssl=false
ApplicationName
```

However, in most realistic situations the connections would be enabled with SSL and password authentication. The following command line can be used to start the application on a specific port.

```
$JAVA_HOME/bin/java -Dcom.sun.management.jmxremote.
port=1097
ApplicationName
```

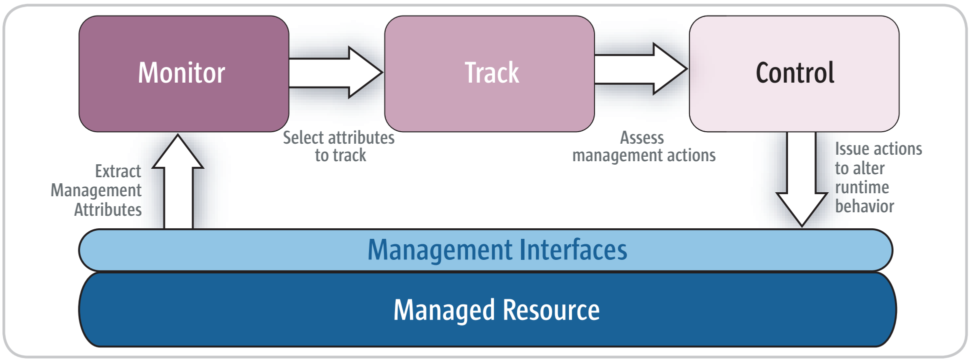

Monitor
Track
Control
Select attributes to track
Assess management actions
Extract Management Attributes
Issue actions to alter runtime behavior
Management Interfaces
Managed Resource

**Figure 1** Application manageability

**Why Settle For "Sorta Close?"**

**Get The BPM That *Fits***

***Reactor***™ *– the ideal Business Process Management (BPM) solution for* ***Workflow Automation, Business Process Integration*** *and* ***Web Services Orchestration***

**Fits Within Your Architecture**

J2EE-based, XML-driven, platform neutral

**Fits Your Business Requirements**

The extensibility your developers want, the simplicity your business users demand

**Fits How You Want To Buy**

Flexibly priced, with source code access

Download your free evaluation copy at www.oakgrovesystems.com, or contact us at 1.818.440.1234. We can't wait to help you…

***Declare Your Workflow Independence!***™

oakgrove systems

Apart from the port number, other configurations need to be set via property files. The JRE comes with property files that are located in $JRE/lib/management/. These files set the various properties required to set up a secure connection when connecting to the JVM. Table 2 outlines the various files required to enable secure out-of-the-box remote management.

*Connecting to the MXBean Interface*

To begin the process of monitoring and tracking, the external entity needs to make a connection to access the MXBean interface. This connection can be made via any supported adapter tool like the SNMP. However, the examples here will be more focused on implementations that use Java. There are two ways to connect: via a proxy or making a connection to the MBeanServer.

A connection can be made by constructing an MXBean proxy instance that forwards the method calls to an MBeanServer (as shown in Listing 1).

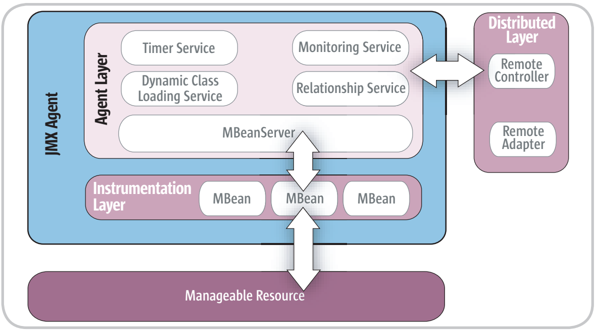


**Figure 2** JMX Architecture

| | |
|---|---|
| **Memory Consumption** | MemoryManagementMXBean<br>MemoryPoolMXBean<br>MemoryMXBean<br>RuntimeMXBean<br>GarbageCollectorMXBean |
| **CPU Usage** | ThreadMXBean<br>OperatingSystemMXBean<br>RuntimeMXBean |

**Table 1** Default management interfaces for memory consumption and CPU usage

| File Name | Description |
|---|---|
| jmxremote.access | The access control file defines the various roles available to access the managed beans. The default roles are the monitor and the control roles. The monitor role gives read-only access to the managed beans. The control role gives read-write access to the managed beans. Other roles can also be defined in this file. |
| jmxremote.password | The password file contains all the passwords for each of the roles defined in the access control file. The default installation comes with a template file (jmxremote.password.template) that can be readily modified by changing the passwords for the monitor and control roles. The passwords are stored in clear text and it's important to ensure that the file has the correct file permissions. The password file can also be located in another location and be specified using the following option at the command line `com.sun.management.jmxremote.password.file =<filename>` |
| management.properties | The management properties define the system properties for SNMP and RMI access. SSL can be enabled by setting `com.sun.management.jmx-remote.ssl=true`. Password authentication can be enabled by setting `com.sun.management.jmxremote.authenticate =true`. |

**Table 2** Security-related property files for out-of-the-box remote monitoring

While the mechanism described in Listing 1 works by getting direct access to an MXBean interface, it's also possible to go through the MBeanServer to get indirect access to the MXBean interface (see Listing 2).

The advantage of connecting through the MBeanServer is that it allows the management application to discover all the MBeans that are registered. This allows the management application to discover a richer set of information in a more dynamic manner. While connecting directly to a specific MBean is simpler, it is more suited to scenarios where the MBeans to be accessed are known beforehand.

After a connection has been established, memory usage for deployed applications can be monitored and tracked by using a polling-based or event-driven mechanism. The JMX architecture supports both of these mechanisms and, depending on the use case, either the polling or the event notification mechanism can be used.

The memory usage can be polled regularly using the getUsage() method. To use the event notification–based mechanism to monitor the memory, the usage threshold can be set using

```
memoryPool.setUsageThreshold(MEMORY_LIMIT);
```

where MEMORY_LIMIT is the peak memory consumption value in bytes.

Setting the usage threshold will generate events of MemoryNotificationInfo when the memory usage exceeds the threshold. When these events are received, additional actions can be taken by IT operators. IT can get detailed information about the state of the application during the time when the application is not behaving properly and this can help developers find the root cause of the problem.

Unusual CPU consumption is one of the indicators that some of the threads in a Java application may not be behaving as expected. The thread-related information available from the ThreadMXBean interface can help provide insights into the underlying problem. The interface not only provides mechanisms to see the number of threads executing, but also provides methods for getting detailed information about each of the threads. The information can indicate the threads that are executing for periods of time that are out of the normal bounds or the threads that are deadlocked. The code in Listing 3 finds if there are any deadlocked threads and prints the information about the blocked and the blocking threads. (Listings 3–7 can be downloaded from www.sys-con.com/java/sourcec.cfm.)

The ThreadInfo class also provides methods to get valuable thread-related information that includes the state of the thread, elapsed time in blocked state, and the stack trace.

One of the important aspects of manageability is the ability to add a degree of control after a certain condition is met. Although the current API doesn't provide controls to tune the JVM, it does provide a mechanism to get additional information when abnormal runtime conditions are observed. This control is provided using the LoggingMXBean interface (see Listing 4). The logging interface allows you to retrieve the various loggers and set the log levels dynamically. This is an extremely important feature because it allows external monitoring entities to acquire detailed runtime information when the application doesn't behave as expected or consumes excessive system resources. It's also important to note that to exercise any kind of runtime control, the management application needs to connect using the authentication for control role.

The new APIs allow the JVM to be managed out-of-the-box by exposing information critical to the health of a JVM. By leveraging JMX as the underlying architecture, it's now possible to externally monitor, track, and control a Java application and this makes Java applications more manageable.

## Internal Monitoring and Management

The monitoring and management APIs can also help create adaptive business logic. Until now it has been difficult to write application logic that takes into account the effects of the runtime environment. Without using the runtime information, it's easy for the application to be in situations in which there are not enough system resources (like memory) available. To make the application more robust, the applications can:

- Expose management information that can be used by IT operators via management software to aid in root cause analysis.
- Take compensatory actions to prevent the application from running into an unrecoverable state.

Although getting information about the JVM can provide insight into the state of the JVM, data exposed at the application level can be equally useful. A developer may choose to expose information at the business-logic level like number of orders pending, the time taken to service requests, and so on.

In this article I look into exposing more granular information at an object level that deals more closely with application logic. For example, collection classes like hashtables and vectors can be easily misused and result in memory leaks. Some of the newer data structures like PriorityBlockingQueue (java.util.concurrent) can also cause OutofMemoryError if not used properly. Based on application logic, information contained in critical classes and utilities like object pools can be exposed via MBeans.

### *Exposing Management Information*

To expose information via MBeans the first step is to create a management interface.

```
public interface ObjectPoolMBean {
 public Integer availableObjects();
 public Integer size();
 public String poolName();

}
```

The ObjectPool class will contain the implementation and the logic that will return the attributes exposed by the management interface (see Listing 5).

When constructing an MBean it's important to ensure that:

- The MBean is a public nonabstract class.
- The MBean has at least one public constructor.
- Standard MBeans implement their own management interface and DynamicMBeans implement the javax.management.DynamicMBean interface.

Once the MBean has been constructed, it needs to be registered with the MBeanServer so that it can be accessed via JMX calls. The following code demonstrates how an MBean can be registered with the default MBeanServer (the PlatformMBeanServer).

```
//Get MBeanServer
MBeanServer platformMBeanserver=ManagementFactory.getPlatformMBeanServer();
//Register the ObjectPool MBean
ObjectName poolName =
new ObjectName("com.foo:id=ObjectPool");
platformMBeanserver.registerMBean
    (new ObjectPool(),poolName);
```

By having the relevant sections of the code exposed as MBeans, it's possible to extract the information about the state of the critical parts of the application. When the application misbehaves, the information exposed via the application-specific MBeans can help narrow the root cause of the problem.

## Taking Compensatory Actions

The exposed management information from an application can alert IT operators about impending problems. The application can also use the JVM information to throttle the application and prevent it from getting into an unrecoverable state. By throttling the application, the garbage collector may get a chance to run and thereby prevent memory or thread-related problems. The example below uses the usage threshold notification mechanism to demonstrate how self-managing logic can be used. In this example, whenever the memory threshold–exceeded notification is received, the application saves the requests so that they can be processed later.

The notification listener (javax.management.NotificationListener) can be extended to detect low memory conditions. The memory threshold notifications are only generated when the memory-usage limits are exceeded. The memory limits are set for the memory pool using setUsageThreshold(). It's also important to note that when a low memory situation continues to occur, additional notifications will not be generated. The next event will be generated only when the memory usage falls below the threshold and then exceeds it again.

**Google Seeks Expert Computer Scientists**

**Google**, the world leader in large-scale information retrieval, is looking for experienced software engineers with superb design and implementation skills and considerable depth and breadth in the areas of high-performance distributed systems, operating systems, data mining, information retrieval, machine learning, and/or related areas. If you have a proven track record based on cutting-edge research and/or large-scale systems development in these areas, we have plenty of challenging projects for you in Mountain View, Santa Monica and New York.

Are you excited about the idea of writing software to process a significant fraction of the world's information in order to make it easily accessible to a significant fraction of the world's population, using one of the world's largest Linux clusters? If so, see **http://www.google.com/cacm**. EOE.

Google

Once the LowMemoryListener has been created (see Listing 6), the listener needs to be registered with the MemoryMXBean. The ManagementFactory is a factory class that provides static methods to get the standard management interfaces for JVM information. The factory can be used to get Memory beans to register the LowMemoryListener.

```
MemoryMXBean membean = ManagementFactory.getMemoryMXBean();
NotificationEmitter emitter
                      = (NotificationEmitter) membean;
LowMemoryListener listener = new MyListener();
emitter.addNotificationListener(listener, null, null);
```

In this example, the saved requests need to be processed at a later time when memory usage is back to the normal level. A polling-based mechanism can be set up to retrieve saved requests and restore the request queue one request at a time (see Listing 7). After the saved request queue is empty, new requests can be accepted.

The example uses the new APIs to build manageability by making the application aware of runtime resource consumption. The application uses a polling-based mechanism to monitor the application state and ensure that all saved requests are processed before new requests are accepted. The memory-exceeded notification is used to track and control application behavior to prevent it from going into an unrecoverable state.

## Guidelines for Exposing Management Data and Taking Adaptive Action

The new APIs enable any application to build manageability by exposing management information. It's important to expose the right type of data to realize the benefits of application manageability. Some of the factors to consider when exposing the management data are:

- ***The granularity of the data:*** The data can be exposed either at an object level or by collating data from multiple objects.
- ***Usefulness and relevancy of the exposed data:*** The information exposed via managed beans has to be helpful to the IT operators from the manageability perspective. A good rule of thumb is that if the information is helpful for debugging and doing a root cause analysis of a problem, it's a worthwhile attribute to expose.

In the example, information about the saved request queue and the JVM memory usage are examples of information that would be helpful to IT operators to predict impending problems in the application.

When adding logic for adaptive business logic for self-management, some of the functional decision points to keep in mind are:

- ***Use runtime information to drive application logic:*** This will allow the application to slow down graciously and thereby allow IT operators to take appropriate actions without incurring application downtime.
- ***Use standard Java logging mechanism:*** This will allow IT operators to retrieve additional diagnostic information via a logging management interface when the application doesn't behave as expected.

## Conclusion

The introduction of the new monitoring and manageability API makes the job of building manageability in applications a lot easier. The JMX support in J2SE 5.0 makes it extremely easy to enable both external and internal monitoring and management. The JMX architecture enables application and JVM attributes to be monitored, tracked, and controlled. Since the JVM can be management-enabled out-of-the box, even existing applications running on older JVMs can be migrated to J2SE 5.0, instantly reaping the benefits of external manageability.

IT operators can now provide detailed information to developers about the application state when the application doesn't behave normally. The JVM information can help developers write self-managing logic that uses the runtime resource consumption characteristics. The new monitoring and manageability API helps developers identify problems quickly, IT operators get detailed information about Java applications and, most important, increases application uptime significantly.

## Further Information

- *Building manageability into software applications:* http://devresource.hp.com/drc/technical_papers/managability-Tech/index.jsp
- *JMX Documentation:* http://java.sun.com/products/JavaManagement/
- *JMX Tutorial:* http://java.sun.com/j2se/1.5.0/docs/guide/jmx/tutorial/tutorialTOC.html
- *WSDM:* http://devresource.hp.com/drc/slide_presentations/wsdm/index.jsp
- *J2SE 5.0 API:* http://java.sun.com/j2se/1.5.0/docs/api/index.html
- *JVM out-of-the-box monitoring:* http://java.sun.com/j2se/1.5.0/docs/guide/management/out-of-the-box.html
- *Jconsole:* http://java.sun.com/j2se/1.5.0/docs/tooldocs/share/jconsole.html

**Listing 1: Connecting using MXBean Proxy**

```
       MBeanServerConnection mBeanServerConnection =
null;
try{

mBeanServerConnection =
                  JMXConnectorFactory.connect(
                  new JMXServiceURL(jmxServiceURL),null)
                        .getMBeanServerConnection();

MemoryMXBean memoryMXBean  =
               ManagementFactory.newPlatformMXBeanProxy(
                        mBeanServerConnection,
                        ManagementFactory.MEMORY_
MXBEAN_NAME,
                        MemoryMXBean.class);

} catch (Exception e) {
            System.err.println("Failed to Connect with "
                    + "JMXServiceURL '" + jmxServiceURL
+
 "':  " + e);

}
```

**Listing 2: Connecting directly to the MBeanServer**

```
// get an instance of the logging MXBean
ObjectInstance loggingInstance =
                mBeanServerConnection.getObjectInstance(
               new ObjectName("java.util.logging:
type=Logging"));

// Use the query mechanism to get all the MXBeans that
are
//part of the java.lang domain.
Set mBeanSet =
          mBeanServerConnection.queryMBeans(
               new ObjectName("java.lang:*"), null);
```

# Ignorance isn't bliss.

## Knowing about performance problems before they happen is.

It might seem like bliss to ignore application performance problems and deal with them only if and when they surface. But waiting until a problem arises in production only results in crises, stress, and wasted time. Experience real bliss by eliminating performance problems proactively with DiagnoSys.

From development through production, DiagnoSys targets potential problems by providing a wide breadth of information, collecting data not only from the application server but also the associated resources, such as the Web server, database, and network.

This breadth of data gives a more accurate picture of a developing problem. Other products only look at the app server and imply performance of the resources based on the performance of the connections, a method that often misses the real source of the problem.

DiagnoSys also provides in-depth, statistically correlated information so you see a more complete view. DiagnoSys gathers information down to the method on the app server as well as in-depth information for all resources down to the OS.

Even though you get depth of information, you won't be overwhelmed. DiagnoSys synthesizes the data and quickly identifies the root cause so you can correct the problem before it impacts you or your company.

diagnosys

**Improve Web app performance at your pace.** Listen to expert Web seminars – it's a fast and easy way to keep up on the latest advice, tips, and best practices.

Visit: www.hwcs.com/jdj05.asp

H&W

www.hwcs.com | 1.800.338.6692

Bridging the Enterprise Computing Gap For 25 Years

# Exploring Enums

*The wait is finally over*

by Ajith Kallambella

To enumerate means to itemize or to list. In the world of programming, enumerations, enums for short, are used to represent a finite set of values (constants) that a variable can attain. In other words, it defines the domain of a type. For instance, different states of a fan switch – off, low, medium, and high – make up an enumeration.

Since the first release of Java, programmers have been complaining about the lack of core language support for enumerated types. After all, Java improvised on the shortcomings of C++ and touted type safety, so it was only natural for the developer community to expect support for a true enum-type. During what seemed like an eternally long wait, many ad-hoc enum representations evolved and most of them shared a common premise – modeling enumerations based on a primitive type, usually an int; see Listing 1 for an example.

Although practical, such implementations have long been frowned upon by Java purists as a hack that can best be described as brittle. What exactly is wrong with such an implementation? In essence, the main disadvantage of faking enumerated types using primitives is the lack of strong typing and hence the inability to catch errors at compile time. Other shortcomings are less readable code, deviations from object-oriented concepts such as encapsulation, the absence of a namespace requiring an explicit prefix for all references, and the dangers of exposing the internal implementation to client code.

Let's look at some of these drawbacks in more detail.

Loose type checking is the result of a compiler treating the faked enumeration just like any other primitive variable. Since the compiler doesn't know anything about the cohesiveness of the enumeration of constant values, it doesn't catch, for instance, a definitive assignment of an out of range value. In other words, lack of an explicit type totally gives away the benefits of compile-time error detection.

Such ad hoc approaches violate some of the basic object-oriented principles. In the previous example, the states of the switch lack encapsulation. Attributes representing the constants and operations are not grouped together but are scattered within the enclosing scope. Since the client code is well aware of implementation details, even a small change such as renumbering them might result in broken client code. Also note the lack of explicit scoping. Since enum attributes are listed along with other class attributes, the enclosing scope is often that of the declaring class or interface. This makes their grouping vague, unreadable, and sometimes error prone.

In his book *Effective Java*, Joshua Bloch presents a pattern for type-safe enums that offers both compile-time safety and better encapsulation. java.awt.Color uses a similar strategy to hide the int-enum implementation under the sheet. However, after reading through pages full of Java code, you begin to wonder if the juice is worth the squeeze. It only reinforces the need for core language support for enum types. For a language that touts type safety, the idea of faking enums with such elaborate pyrotechnics seems rather odd.

## Enter the New Enum

The wait is finally over and true enums are here. Enums are a type of their own in J2SE 5.0. Among many other developer-friendly features recommended in JSR 176, Tiger (the code name for the new release) packs the powerful punch of type-safe enums. Listing 2 shows how we rewrite Fan.java using true enums. (Listings 2–9 can be downloaded from www.sys-con.com/java/sourcec.cfm.)

Note the new enum keyword introduced to support the new type and also how enumerated constants are listed – they are neither strings nor ints but belong to their own *declared* type, i.e., SwitchState. Because it's a true type, all the benefits of strong typing automatically kick in. For instance, the following code attempting to set an invalid state

```
new Fan().setState(6);
```

is automatically detected at compile time and reported an error, eliminating the need for several lines of validation code:

```
setState(Fan.SwitchState) in Fan cannot be
applied to (int)
```

Let's look at other useful features provided by the enum type.

### Support for Namespace

In the previous example, all the listed constants, e.g., states of the fan switch, implicitly belong to the enclosing type, appropriately named SwitchState, requiring an explicit namespace qualification to all references to states. For example:

```
 SwitchState.Low
```

Namespace adornment also helps avoid collisions. If we were to declare another enum type for class Fan using a similar set of constants, say:

```
 public enum DurabilityRating { Low, Medium,
High }
```

it will result in no name collision. SwitchState.Low can be distinguished from DurabilityRating.Low because of their enclosing scope.

Since enum SwitchState is declared as public, and with the type system providing a namespace for each enum constant, it can be accessed outside the scope of class Fan just like any normal attribute reference, e.g., Fan.SwitchState.Medium.

### Auto Conversion to String Values

The new enum type easily facilitates descriptive printing using an internal implementation of the toString() method. The default implementation returns a string representation of the constant as declared, and, if you don't like it, it can be changed by overriding toString(). It's that simple.

**Ajith Kallambella** is a Sun Certified Enterprise Architect and holds several other certifications from Sun and IBM. With over 10 years of industry experience, he has been developing software for the distributed enterprise using J2EE since its very first release. Ajith is co-author of *Java 2 Certification Passport* and a moderator (sheriff) of Java-related discussion forums at JavaRanch.com. He works as a senior computer scientist for Computer Sciences Corporation where he architects scalable client/server solutions.

*ajith@javaranch.com*

*Works with Programming Constructs*

Since enums are their own types, some of the standard Java constructs, notably the switch and for statements, have been enhanced to work with enum types. Listing 3 provides an example of the switch construct.

Switch statements are useful for simulating the addition of a method to an enum type from outside the type. This can be very useful if for some reason the enum definition cannot be modified, but needs to be extended.

There is something rather interesting about this switch statement – note how the enum constants in case statements appear bare, e.g., unqualified with their namespace. J2SE 5.0 makes life easier by attempting to resolve identifiers in the immediate context. This is somewhat similar to the new static imports feature. In this example, however, you are required to omit the namespace. Try including it and you'll get an error message:

```
Fan.java:16: an enum switch case label must be the unqualified name of an enu-
meration constant
                 case SwitchState.Off : return SwitchState.Low ;
```

Listing 4 provides an example that uses a for-loop construct to iterate through enums.

The values() method returns an array that contains enum constants for this enum in the order in which they are declared. By the way, the for-loop shown here is called "enhanced for-loop", another neat feature in the new release. They are also called as "for-in loop" since you read them as "for-state-in-Switchstate.values". This new construct simplifies iterating over a collection of values – both arrays and Java collections – by taking away the need to inspect the size, use a temporary index variable, and cast each element to the appropriate type.

## Looking Under the Hood

In the spirit of empiricism, let's look under the hood and see how enums are implemented. Compiling Fan.java (for the complete source code see the resources section) generates two .classes: Fan.class and Fan$SwitchState.class. The latter contains our enum definition.

```
D:\MyJava\JDJ\src>javap  Fan$SwitchState

Compiled from "Fan.java"
public final class Fan$SwitchState extends java.lang.Enum{
    public static final Fan$SwitchState Off;
    public static final Fan$SwitchState Low;
    public static final Fan$SwitchState Medium;
    public static final Fan$SwitchState High;
    public static final Fan$SwitchState[] values();
    public static Fan$SwitchState valueOf(java.lang.String);
    static {};
}
```

As you can see, the process of compilation results in the automatic generation of a new class that extends java.lang.Enum. In other words, the enum keyword acts as a shorthand representation for the autogenerated class. This is what the authors of JSR 201 meant by "linguistic support for type-safe enumeration pattern." It's worth mentioning that every new language extension introduced in Java 5.0 is implemented by modifying the source-to-byte code compiler so the JVM implementation remains unchanged.

If you notice the naming convention adopted for generated enum types, you'll recognize that they closely resemble the inner class syntax, e.g., <enclosingType>$<thisType> format. All enum classes are final subclasses of java.lang.Enum, serializable, and comparable. They come with predefined toString, hashCode, and equals methods. All methods except toString are final.

Note that all enlisted enum constants are represented as final self-type variables of the class. This is done to ensure no instances exist other than those in the generated class. In other words, enums are Singletons and, for the same reason, they can't be instantiated using new or cloned. The clone method in java.lang.Enum throws a CloneNotSupportedException.

Since enums are compiled into their own class files, their definition can be changed, e.g., enum constants can be added, deleted, and reordered without having to recompile the clients. If you removed an enum constant that a client is using, you'll fail fast with an error message.

Now let's look at some advanced features.

*Flavors of Declaration*

Enums can be declared in various flavors. The enum FanState above is an example of an inline declaration. These are useful for defining enums that have a limited scope – the use and throw type of enums. Since Java 5.0 introduces the keyword enum at the same level as class and interface keywords, enums can be declared just as you would a new class or an interface – in its own .java file (see Listing 5).

Normal rules of access visibility that apply to standalone Java classes also apply to enums, whether they are declared inline or in a separate file. For example, if you omitted the public keyword, the enum type will have the default package visibility and hence be accessible only within the package. Similarly if the SwitchState enum were to have a private access specifier, it wouldn't be accessible outside the class scope of Fan. You get the point.

As with other new features introduced in Java 5.0 such as generics, enums have been used in several JDK packages. JDK 5.0 includes several enum types to support core classes. For instance, the newly introduced java.lang.Thread class uses an enum named State to represent the current state of the thread. Some of the best enum candidates haven't been converted over yet – like the most quoted Color class. You can use Javadoc

Code hard.

**Play hard.**

Why ***wouldn't*** you want to work at Qpass? Join the leader in m-commerce software.

Serious (and seriously fun) Java developers can learn more at **www.qpass.com/jdj**

QPASS®

as your field guide for spotting enums in Tigerland. The new API Javadoc conveniently lists all enums in the package-summary.html along with other top-level types. Once you have spotted them, go ahead and open the source code and see how the preachers practice. It's always fun.

## Class-like Behavior

Since enums are class-like critters, they support most, if not all (see the Caveat Emptor sidebar), semantics supported by normal Java classes. The enum type definition can have one or more parameterized constructors, implement interfaces, and support class body elements such as attributes, methods, instance and static initializer blocks, and even inner classes.

In addition, arbitrary fields and methods can be added to individual enum constants. Such constant-specific class bodies define anonymous classes inside enum classes that extend the enclosing enum class. Listings 6–9 illustrate the use of some of these features.

There's a lot to digest here, so let's tackle them one at a time.

We are defining three types of accounts as enums – checking, savings, and investment. First things first – we need some clarification of terminology here. Every enum constant has a declaring class, which is also called its type. The term "enum type" is used to refer to the declaring class, e.g., AccountType. When we use the term "enum constant", it's referring to all enlisted contents contained in the scope of AccountType, e.g., Checking, Savings, and Investment. It shouldn't be very confusing. Just think of any class declaration and its objects – the enum type is analogous to the class declaration and enum constants, the objects.

### *Enums Are Classes*

Similarities between a normal class declaration and an enum declaration are hard to miss. The enum type AccountType implements the IAccountType interface and has two constructors. What does it mean? To implement an interface, the enum type must implement every method defined in the interface. Since all enum constants are objects of the declared type, they share the common implementation of methods getAvgBalanceMethod and getInterestRate. An enum constant declaration, when followed by arguments, invokes the constructor defined by the enum type. In our example, the enum constant Savings invokes the constructor

```
AccountType(double interestRate)
```

with argument 3.5.

All the standard rules of constructor overloading and selection specified by the Java Language Specification are followed here. Since enum constant Checking has no arguments following its declaration, it's necessary to provide a no-arg default constructor. If the enum class has no constructor declarations, a parameterless default constructor is automatically provided to match the implicit empty argument list.

Notice how an enum type can declare methods and attributes, again, just like a normal Java class. For member type declaration, all rules of scoping, access specifiers, and visibility are valid and therefore must be followed with regard to instance variables and methods. There is something rather interesting with the type AvgBalanceMethod. It's an enum within an enum. Since an enum type can support all types of class members, they can support enums too. Since AvgBalanceMethod is declared as public, it's visible outside the scope of the enclosing AccountType. Notice how it's accessed in the Account class.

### *Constant Class Bodies*

Enum constants can be associated with an arbitrary class body often referred to as a constant class body. Such classes are very similar to anonymous class declarations (even named similarly) and are implicitly static. This means they can access only static defined in the enclosing scope. Since constant classes are implicit extensions of the enclosing class, they can override methods defined in the enum type. In our example, enum constant Investment overrides getAvgBalanceMethod defined by AccountType. In fact, it would be perfectly legal to declare AccountType as abstract and force every enlisted enum constant to implement methods in the interface in their constant class body. A

## Caveat Emptor

Let the buyer beware: a simple axiom often used in commerce summarizes it well – it means the buyer alone is responsible for assessing the quality of a purchase before buying. Enums are so powerful, once you learn the ropes it's easy to be tempted to stretch and do "cool" things. Before you know it, bad things start happening.

Always check out the API spec and know the expected behavior before you put something to use. As a smart programmer, you also need to know the limitations of the enums so that it can save your bacon some day.

1. *You can't new enums:* Remember they are singletons. For the JVM to handle them properly, only one instance of each should exist. They are automatically created for you. Luckily the compiler catches explicit instantiation. For the same reason, enum constructors are implicitly private. Think about it.
2. *You can't extend enums:* One enum cannot extend another one. You can't extend the primordial java.lang.Enum either. That's how it works and if you are worried about this limitation, your design demands a second look. You shouldn't need a hierarchy of enums.
3. *You can't declare enums locally:* Enum types cannot exist in any scope lower than a class scope. You can't define them within a method.
4. *Order matters:* Within the enum type class body, the constants must appear before other class elements such as attributes, methods, and constructors. This is true at least as of the 5.0 beta release.
5. *No nulls please:* An enum constant cannot be null. It's that simple.
6. *Don't use ordinal:* Code that uses an ordinal() method, e.g., logic based on the position of an enum constant as it appears in the declaration, must be discouraged. Your code will break at runtime if constants are subsequently reordered. What's more, this is a runtime error. The compiler will not catch such a thing.
7. *Everything that sounds the same, isn't:* There are a few other classes in JDK 5.0 that sound very similar – Enumeration, EnumControl, to name a few. Don't assume they are enum implementations. Check the Javadoc.
8. *Spare the serialization:* Enum serialization isn't like the normal one you have seen. The process by which enum constants are serialized cannot be customized. Any class-specific writeObject and writeReplace methods defined by enum types are ignored during serialization. Similarly, any serialPersistentFields or serialVersionUID field declarations are also ignored – all enum types have a fixed serialVersionUID of 0L. Again, this shouldn't concern you too much. Let the language take care of the specifics.

**Listing 1: Enumerated types – the old style**

```
package simple;

public class FanOldStyle {

    public static final int SwitchState_off = 0 ;
    public static final int SwitchState_low = 1 ;
    public static final int SwitchState_medium = 2 ;
    public static final int SwitchState_high = 3 ;

    private int currentState = SwitchState_off;

    public void setState(int state){
    // Out of range argument can result in illegal state.
        currentState = state ;
    }

}
```

word of caution: although support for constant class bodies is a very powerful feature, it may be wise to avoid stretching them because of the same reasons why excessive use of anonymous classes is discouraged – they are less readable and hard to debug.

It's important to mention here that only a nonfinal instance method in java.lang.Enum is the toString() method. You can override and implement the per-constant toString() method to return a descriptive name for each enum constant. The default implementation of toString() returns just the string equivalent of the constant. Therefore, as illustrated in the example, it may be a good idea to override when a more descriptive literal is necessary.

Before we proclaim victory, two new classes introduced for enum support deserve consideration: java.util.EnumMap and java.util.EnumSet. As their names suggest, they are enum-enabled counterparts of the standard Java collection implementation. They both require that each element maintained by the collection belong to one enum type. In short, they won't let you mix and match enum constants from different types. It's rather interesting that they both retain the elements, e.g., the enum constants, in their declared order, totally ignoring any custom implementation of the compareTo() method. Be sure to check out their API documentation.

## Conclusion

The new enums are a robust way to implement constant named lists and make them a compelling alternative to ad hoc enum implementations. With class-like behavior and the ability to support arbitrary class members, they are certainly bigger than they appear. Although you may not have an immediate need to use them in your projects, or at least not every feature they offer, keep them in your toolbox and you'll soon find them handy.

## Resources

- *Java 5.0 Release candidate home page:* http://java.sun.com/j2se/1.5.0 index.jsp
- *JSR 201 – check out the enum draft spec:* http://jcp.org/en/jsr/detail? id=201
- *JSR 176 – J2SE 5.0 Release contents:* http://jcp.org/en/jsr/detail? id=176

Technology is hot again. Is your career? **NOW** is the time to explore new opportunities.

Visit Dice.com to find a better job with better pay. Check your salary. Compare your skills. Search over 50,000 tech jobs from leading companies and choose to have new jobs emailed to you daily.

**IT'S TIME** for something better.
***Visit Dice.com today.***

**Dice**™
***Look to the tech leader first.***™

Joe Winchester
Desktop Java Editor

# Square Data and Round Holes

My first programming job was done using Report Generator Language (RPG) on the IBM System 36. The hardware was green screen, the tape decks reel-to-reel, and the printers large and noisy. The language itself was very data-centric with each program declaring formatted Input or Output data structures that were read or written to. Each structure mapped to a file, a screen buffer, or a printer spool. In spite of all this we did get the job done, although our biggest problem was the business changing requirements on us that necessitated altering the data structures. Because they were burned into the program's source specification, changing a file required altering every program that used it and a conversion job to update the live user data to use the new format. The inertia of this task made it important to do thorough up-front analysis to get the data relationships and attributes as correct as the available knowledge allowed.

Preemptive flexibility sometimes included soft-coding all of the data structures to be called anonymous names such as "user field 1" or "user field 2." Separate definition files for each application mapped which fields were used in which context and the intention was that when the business required some new data attribute to be added, an unused extra structure was simply activated by hacking around with the definition files to make the new attribute available on screens and reports.

The thing that grabbed me about object-oriented programming when I was first introduced to it (through the Smalltalk language and then Java) was that all of this would be fixed. The program was no longer concerned with its data; instead this was all encapsulated inside objects that provided a behavioral API, making the system more malleable and extensible. Inheritance and polymorphism and other facets are nice features of the language, but data encapsulation was the key thing that sold me.

The first couple of systems I worked on were business apps that had to deal with back-end relational corporate data, and the problem that arose is the well-trodden one of how to persist objects in a relational database. Objects have things like inheritance, many-to-many relationships, many ended links with no back pointer, untyped data structures (in the case of Smalltalk), and other facets that just don't fit into a row/column fashion. Initially as I wrestled with this impedance mismatch by writing or using fancy frameworks, I always believed that this was a temporary point-in-time exercise required because of the existence of legacy relational databases; OO databases were just around the corner so their arrival would cure all our maladies.

Frederick Brooks has a chapter "No Silver Bullet" in his superb treatise *The Mythical Man Month*, and unfortunately my belief in OO databases fell naively foul of his prediction. I had the good fortune to work with a very powerful OO database while programming for a bank. It did effectively persist the objects, however, it failed to meet the business' needs precisely because it was structured around objects and not rows and columns. Users needed fast and varied access to their data, and just about every existing application from spreadsheets to off-the-shelf GUI builder tools was on their desktops itching to access the corporate data. These all required the data to be relational, and although an ODBC-to-OO bridge existed, apart from being some kind of intellectual nasty bolt on, it meant that the user's thinking of the data was by definition a relational one. After a while some converts began suggesting it was pointless to be doing OO at all, and one strong argument came from the fact that the data itself was inherently row/column based because that was how it was received by the system and its users. Input came from external data feeds (where the data was structured) or from manual input where tables and lists were rows of data, and fields of data populated the columns.

Object-oriented databases do work and are widely used in apps that don't need to publish their data as a corporate database (such as embedded devices); however, for the corporate world it perhaps looks as though OO databases haven't achieved the critical mass required to become widely accepted. On one major database vendor's Web site, the listing of their product portfolio promoted their relational mapping software in preference to their (very good) OO database as the initial deployment configuration for J2EE.

One way in which object-oriented data stores might enjoy a renaissance is with XML. By its nature XML is structured around a tree of nodes that can repeat and contain further nodes, data elements are optional, and, although XML enjoys being used as a readable structured message format, it's also used as a way of representing and persisting data. XML doesn't store itself in rows and columns easily; however, if it's persisted in raw text for queries a search engine needs to be able to peek at its contents. This is essentially what Web search engines do – initially they just queried into HTML (which is no more than XML marked up specifically for browser syntax), although now they recognize specific content formats (.doc, .pdf) and promise to even embrace the desktop's contents itself as their data source (http://news.com.com/Google+to+unveil+desktop+search/2100-1024_3-5408765.html?tag=nefd.lede).

If the future of search engines is to query data irrespective of source, and the flexible and user friendly nature of searches exceeds anything that SQL could do for a nontechnical corporate user, is it possible that object-oriented databases will be reborn with the required search engine interfaces? Or is the problem simply that data sticks where it lands, and most companies are loathe to physically move data from its initial resting place lest the downtime and potential errors create more problems than are solved?

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

*joewinchester@sys-con.com*

## Is your SOA really agile?

Building, testing and maintaining SOA solutions requires a flexible and collaborative approach to diagnostics. Finger pointing and confusion occur when all parties do not have a complete and common understanding of a problem. Mindreef® SOAPscope® 4.0 connects developers, testers, support personnel, and operations by combining the testing and diagnostic tools for each discipline and making it easy to share complete problem data. SOAPscope is the only end-to-end diagnostic system for Web services.

## Developers • Testers • Operations • Support

### NEW! SOAPscope 4.0

**Test** - Agile testing approach for both developers and testers allows what-if scenarios without coding or scripting. Includes the industry's most complete interoperability suite and the flexibility to really test the edge cases.

**Capture** - Easily capture and assemble complete problem data into SOAPscope's workspace. Capture SOAP messages off the wire, test cases, or re-produced problems. Capture a snapshot in time of a WSDL including all related resources.

**Share** - Exported workspaces support sharing among team members preserving complete problem data including messages, WSDLs, and notes. Sharing workspaces is a great way to support web service users.

**Solve** - Diagnostic suite of analysis, comparison and visibility tools simplifies the understanding of complex XML, yet are powerful enough for the industry's most protocol-savvy experts.

**"SOAPscope has been invaluable in testing and diagnosing our enterprise scale Web Services. The new Workspace feature in SOAPscope 4.0 is an excellent way to support customers and collaborate as a team. A must have for every WS-Developer"**

*Simon Fell*
*Senior Member of Technical Staff at salesforce.com and author of PocketSOAP*

Try SOAPscope **FREE** at www.Mindreef.com

Mindreef, Inc.
www.Mindreef.com

# JAVA GAMING

## 2D rendering PART 2

by Chet Haase
and Dmitri Trembovetski

*Part 1 of this article ("Java Gaming: Understanding the Basic Concepts," [**JDJ**, Vol. 9, issue 10]) covered the basics of a game framework. Part 2 goes into more depth on the actual 2D rendering specifics and the resulting demo: the Ping program (see Figure 1).*

## 2D Rendering

Game rendering is a subject that has great depth and complexity. This article focuses on the topics that we believe are the most important to 2D games and Java games programmers:

- Fullscreen and DisplayMode management
- Buffering
- Images
- Video memory constraints
- Performance tip: intermediate images

## Fullscreen and DisplayMode Management

A game developer must decide whether to run a game in fullscreen mode (where it occupies the entire monitor display) or windowed mode (where it is one of many windows on the user's desktop). Both modes are appropriate for different types of games. For example, a game that is supposed to be all-consuming while being played would work best in fullscreen mode, allowing it to take over the machine and let the user be completely mesmerized by the experience. Alternatively, a game that is fun to dive into and out of briefly, such as a card game, may best be one of many tasks on a user's monitor (allowing them to pretend to actually get real work done at the same time).

Another approach is to be flexible and allow the game to run in fullscreen mode or in windowed mode. This property could be user configurable and the game written to work well either way.

### *Fullscreen Mode*

To put your game into fullscreen mode first create a Frame, make it undecorated (this removes the window decorations such as the title bar and close/resize icons), and then tell the appropriate GraphicsDevice to switch into fullscreen mode on that Frame:

```
Frame gf;
void initFullScreen(GraphicsDevice gd) {
  // initialize the main app frame
  gf = new Frame("Game Frame");
  gf.setUndecorated(true);
  // disable repaint mechanism
  gf.setIgnoreRepaint(true);
  // the next call shows the window
  gd.setFullScreenWindow(gf);
 }
```

The call to setIgnoreRepaint() is used because the game loop described in the first part of the article handles all rendering explicitly, meaning we don't need the usual toolkit mechanism of repainting the window.

### *DisplayMode*

If your game chooses to run in fullscreen mode, you also have the ability to switch the DisplayMode or resolution of the monitor. This is platform dependent and currently exists only on Windows, although we hope to have this capability available for other platforms in the future. The following code is an example of how to set the DisplayMode.

```
if (gd.isDisplayChangeSupported()) {
  DisplayMode myDM = new DisplayMode(640, 480, 32, 60);
  try {
    gd.setDisplayMode(myDM);
  } catch (IllegalArgumentException iae) {
    DisplayMode dms[] = gd.getDisplayModes();
    for (DisplayMode dm : dms) {
      if (dm.getWidth() == 640 &&
          dm.getHeight() == 480)
      {
        gd.setDisplayMode(dm);
         return;
      }
    }
  }
}
```

This code could be used in the initFullScreen() method to set the current resolution to 640 pixels wide by 480 pixels high with 32 bits per pixel color depth and 60 frames per second refresh rate. Appropriate error checking is used, beginning with a check that the system can handle switching display modes (some platforms don't currently allow this; others allow it only when running in fullscreen mode). Then, if the program

**Chet Haase** is an engineer in the Java 2D team at Sun Microsystems, Inc., where he focuses on graphics rendering and performance issues for the Microsoft Windows platform. Catch his blogs and articles at http://javadesktop.org.

*chet.haase@sun.com*

**Dmitri Trembovetski** is an engineer in the Java 2D team at Sun Microsystems, Inc., where he focuses on graphics rendering and performance issues for the Solaris and Linux platforms. Read his frequent posts on the forums at http://javagaming.org.

*dmitri.trembovetski@sun.com*

cannot set the exact display mode that it would prefer, it iterates over the possibilities, looking for a match that is close enough.

## Buffering

Games, or any applications that want fast and smooth animation, should use buffering instead of drawing directly to the screen. Drawing to the screen works fine in cases such as simple and static GUIs; however, for constantly changing graphics, rendering each item to the screen causes flashing artifacts that are disturbing to the user. It's far better to render all objects to a back buffer and then copy the buffer to the screen all at once. This technique is called double buffering and makes for much smoother animations and more enveloping game experiences.

There are several approaches game developers can take in creating a double-buffered application.

### *Use Swing*

Swing already has a back buffer built in. When the program renders to the Graphics object passed to an overridden paintComponent() method in a Swing component subclass, it's usually rendering to the Swing back buffer. This buffer is eventually copied to the screen. You as a developer do not have to worry about the details of this buffer; your program renders things into the right places and Swing takes care of the rest.

This approach works well for many simple games, but for games that want full control over all of the rendering aspects (such as those that use their own rendering loop), leaving the buffering under the control of other code such as Swing may not be sufficient. These games may want to control exactly when that buffer is rendered to and copied to the screen. For example, to achieve a consistent frame rate it's necessary to control exactly when rendering and buffering occur. For these applications Swing's double buffering is not the answer.

### *Manual Buffer Image Creation and Management*

The traditional way to do double buffering manually is to create an offscreen image, render to that image in the main game loop, and then call Graphics.drawImage() from that image to the screen Graphics when the frame rendering is complete.

In JDK 1.4, the new twist to this approach was the introduction of VolatileImage, which made it possible to create the offscreen image in hardware-accelerated memory on the graphics device.

### *BufferStrategy*

Also in JDK 1.4, the new API of BufferStrategy was introduced. This API boils down the actual work in managing buffers into the basics that a developer needs, allowing the program to create a back buffer (which can be either a Flip or Blt buffer, discussed below), get the Graphics for that buffer, render to the Graphics, and then tell the buffer to show() itself. These buffers may get hardware accelerated (by using VolatileImage under the hood) without the hassles inherent in managing VolatileImages. Moreover, the ability for BufferStrategy to use the most appropriate back buffer (Flip or Blt) means that you always use the same API and let the BufferStrategy implementation take care of the details.

Creating and using a BufferStrategy is shown below. Assuming the same gf Frame variable used in initFullScreen() above, the program can initialize the BufferStrategy as follows:

```
gf.createBufferStrategy(2);
// 2 means one back buffer and one screen (2 total)
BufferStrategy strategy = gf.getBufferStrategy();
```

and use it later in the rendering process:

```
Graphics g = strategy.getDrawGraphics();
// render to the Graphics object appropriately
// ...
// now show the back buffer on the screen
strategy.show();
```

### *Flipping and Blitting Buffers*

When an application is running in fullscreen mode on some platforms, createBufferStrategy may create a FlipBufferStrategy. Otherwise, a BltBufferStrategy will be used. FlipBufferStrategy will get the buffer contents onto the screen by a simple pointer switch (swapping the pointers for the back buffer and the screen memory). The BltBufferStrategy will always copy the contents onto the screen (by calling Graphics.drawImage() internally).

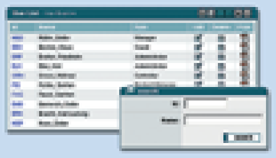
Buffer flipping is an inherently faster operation than buffer copying since it requires only a pointer swap. However, flipping will usually wait for the next vertical refresh event to occur. This means that you may actually get a slower frame rate overall even though the actual flip operation can happen very quickly. This slowdown is because you are now pegged at a maximum of the refresh rate of the video card (60 times per second, or whatever it is set at).

Another implication of buffer flipping is that your application will have smoother animations in general because the buffers are swapped instantly at a time when there are no other changes on the screen (because the flip happens between vertical refreshes). Imagine if the screen was in mid-repaint and was halfway through drawing an object that is currently moving in your game. If you flip buffers, the refresh will finish before the buffers swap, thus that whole frame remains consistent to itself. If you copy from the back buffer to the screen while the refresh is happening (as would be the case with BltBufferStrategy), you might get a "tearing" artifact where that moving object is seen with its top half in the previous location and its bottom half in the new location during the same screen refresh. This tearing artifact is seen in Figures 2–4.

COMMON CONTROLS www.common-controls.com

The Java™ Presentation framework for J2EE™ Web applications

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



For Free!

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

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

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

www.common-controls.com

In general, we advise using BufferStrategy for game programming; it gives game developers the hands-on control they desire while taking care of many of the tedious details of buffer management for you.

## Images

There are mainly two types of images used heavily in games.

### *Sprites*

Sprites are images that are rendered to seldom or never (perhaps they are loaded or created/rendered once at startup and never touched again), but are copied from quite often, perhaps once or more per frame. Examples include icons in a GUI or player character images.

### *Backbuffer*

This is an image that is rendered to often (say, several times per frame) and copied from once per frame.

In Java, the best images to use in these cases are:

- ***Managed images for sprites:*** These are images whose main copy is stored in the Java heap (such as a BufferedImage), but for which we might create a cached accelerated copy in video memory. Although operations to the image are not hardware-accelerated (because we don't enable hardware acceleration for rendering to Java heap-based images), it is the copies from the images that you really care about (since these are the operations that occur over and over) and we will, if possible, accelerate those operations.
- ***VolatileImage or BufferStrategy for Backbuffer:*** Only by using these APIs can you create a back buffer that is capable of accelerating rendering both to and from the image.

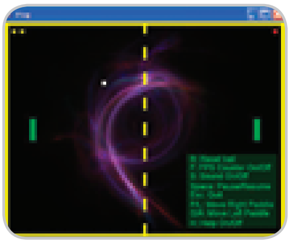


**Figure 1** Ping demo program: application and source code at http://ping.dev.java.net

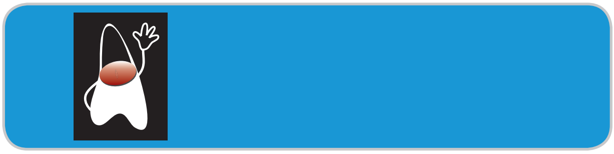


**Figure 2** Frame 1, with the actual position of Duke on the screen

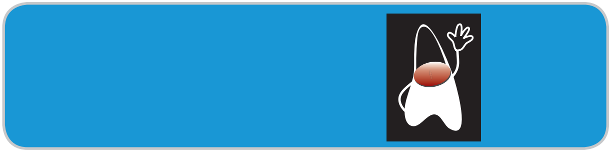


**Figure 3** Frame 2, with Duke in the new position

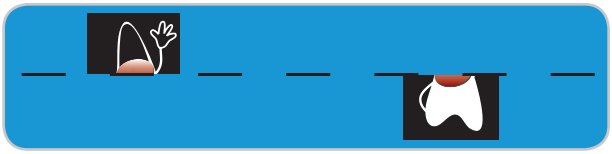


**Figure 4** Tearing artifact seen when copying Frame 2 onto the screen while the vertical refresh (whose position is indicated by the dashed line) is in the same area being updated

### *ImageIO*

Note also that ImageIO is a good package to keep in mind for general image loading and saving. This package was new in 1.4 and was created to be a more general purpose and robust image reading/writing facility than the old image-loading APIs of previous releases. The following are some of the reasons to consider using ImageIO for your future applications.

### *Synchronous*

The old image APIs assumed that you wanted to load your images asynchronously and then deal with them when they were loaded. This is still a valid way to go for many uses, but not all, and people end up forcing synchronization by either using the MediaTracker API or by using ImageIcon (which is merely a Swing wrapper around the Toolkit image facility coupled with MediaTracker).

### *More Image Formats*

There are currently more image formats supported in ImageIO, and any future work we do for further image formats is expected to happen only for the ImageIO APIs. The currently supported formats in ImageIO readers include JPEG, BMP, GIF (read only), WBMP, and PNG.

### *Pluggable Reader/Writer API*

If you have some other custom image format that you need to use, you can write a plugin for ImageIO instead of writing the entire image loading/saving framework from scratch.

### *More Hands-on Capability*

The old image APIs did not encourage image manipulation; you could load an image, but you couldn't really get at the bits (either the pixel data or the metadata) very easily. ImageIO supports both easier pixel access (it creates BufferedImage objects) as well as metadata access. More control equals more power, and more power is a wonderful thing.

### *All Images Are Modifiable*

The old image APIs created images that were read-only. The ImageIO images (BufferedImage objects) are all writeable.

Note too that ImageIO images (in addition to all other image types) are "managed" in JDK 5.0; we'll automatically accelerate copies from these images as described above under the "Managed Images for Sprites" discussion.

Loading an image using ImageIO can be done with the following snippet:

```
BufferedImage bi = ImageIO.read(new File("Duke.png"));
```

## Video Memory Management

Accelerated video memory (VRAM) can be a very constrained resource on some machines, especially ones with low-end graphics cards, such as most laptops. When a program creates images to live in video memory, it's using up that scarce resource in a way that may force other, more important, images to live in system memory instead.

As with any scarce resource, it's important to manage VRAM carefully, especially with a performance-oriented application that benefits from having exactly the right images accelerated at exactly the right times.

Currently, VRAM must be tracked manually using simple methods to inquire about the availability of VRAM:

```
GraphicsDevice.getAvailableAcceleratedMemory();
```

and force images to dispose of any associated VRAM resources:

```
Image.flush();
```

In addition, you can use the ImageCapabilities API to determine whether any given image is accelerated:

```
Image.getCapabilities(GraphicsConfiguration gc);
ImageCapabilities.isAccelerated();
```

## Performance Tip: Intermediate Images

We will leave you with one general performance tip for 2D rendering: if you ever need to render anything even mildly complex several times, consider prerendering it as an image first and then simply calling drawImage() from that image from then on. Consider the following examples.

*Transformed Images*

Suppose you have an original image (say, a sprite) that you want to transform (scale, rotate, whatever) prior to rendering, and then you intend to render it in this transformed way several times. It's probably much faster to create an intermediate image to hold the transformed result and then simply call drawImage() from that intermediate image than it is to make us transform the original image every time you render it.

*Text*

Currently we render text through software routines (except in our new, cool OpenGL rendering pipeline, available in JDK 5.0 but disabled by default). We'll eventually cache text characters (glyphs) as accelerated images, but you could do the same in the meantime. In fact, you could do even better than that: you could cache entire strings as images. Say, for example, you want to display the string "Score:" at the top of the screen on every frame. It's always the same text, in the same font, at the same size, and in the same color. Why not create an intermediate image, render the text to that image, and then call drawImage() from that intermediate image from then on? It's far faster for us to copy from that image than to go through the work of rendering the actual characters every time. As if these advantages were not enough, you can take the same approach with such text variations as custom fonts, anti-aliased text, and large font sizes, all without any performance penalty that you might currently experience by using our default text rendering; once it's an image, all we need to do is copy it.

*Anything*

It's easy to see that the above approach of rendering transformed images or text into intermediate images could be taken with any kind of rendering whatsoever. A snowflake that consists of a lot of lines? A complex Shape that takes a long time to render? An icon you copy around often? All you need to do is create the image, get the Graphics for it, render your object to the image, and then call drawImage thereafter whenever you would have otherwise called the actual rendering operations.

There are (of course) a couple of important details in this approach that we should mention.

*Image Type*

You'll need to create an image of the appropriate type for your desired objective. For example, if you are using an intermediate image to render text, you'll need a transparent image so that the background of that image does not get copied along with the text characters. Similarly, if you are using an intermediate image for a rotated version of your image, you probably want the area outside that rotated image to be transparent. If you are using an intermediate image to do antialiasing or translucent rendering, you'll obviously want a translucent image. Note that copies from translucent images may be much slower than copies from opaque or transparent images, so you may want to test the alternatives to determine what will work best for your situation.

*Size vs Speed*

The only downside to using intermediate images is that you are trading potentially faster performance for an increased memory footprint; every one of those additional intermediate images consumes a chunk of memory. You need to determine whether that increased memory usage is worth the trade-off of better performance.

## Intermediate Images: For More Information

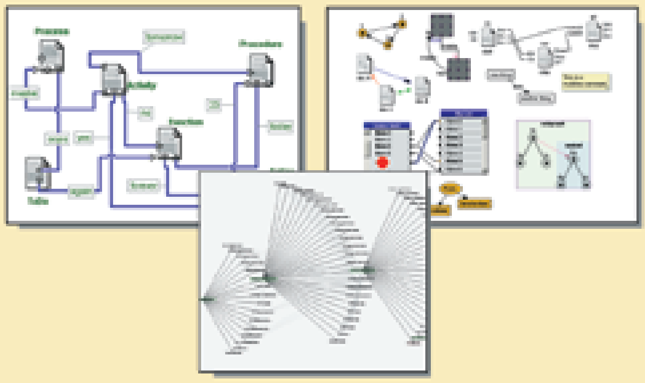
This topic is covered in more depth with sample code on Sun's developer site; check out the article "Intermediate Images" at http:// java.sun.com/developer/technicalArticles/Media/intimages.

## Demo: Ping

The source code and related information for this demo can be found at http://ping.dev.java.net.

We wrote the Ping application to demonstrate points that we were trying to make around the game framework described in Part 1 and general 2D game programming issues and performance tips. It's not supposed to be a game that lasts you through months of sleepless nights (or even through a half hour of twiddling). It should, however, be a fairly complete example of

**Build Incredible Interactive Diagrams with JGo™**



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

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

*Learn more at:*
**www.nwoods.com/go**
**800-434-9820**

Northwoods
SOFTWARE CORPORATION

a simple game framework and should have enough sample code and algorithms in it to show how you might use similar approaches in a game that is more complete and awesome. In short, we expect you to spend far more time with the Ping code than you will playing the actual Ping game. We hope that you can use the code and approaches in it to create something truly great.

The code should stand pretty much on its own, especially given that Ping uses the game framework and 2D rendering tips described in this two-part article. However, as with any application that takes more than 50 lines of source code, a little high-level explanation would probably be helpful. The next section will introduce the game and describe the architecture at a high level. It will also call out methods and approaches of particular interest, especially in the context of this article. However, we don't intend to cover everything of interest in the demo here; we would suggest you dive into the code to see for yourself how it all works.

## Game Description

I can't imagine anyone who has read this far into the article who has not seen a game similar to Ping, so a description of the game is probably redundant. But in the interests of completeness…

Ping is a simple 2D version of tennis, where the ball bounces around the playing area. The object of each player (represented by paddles on the left and right) is to keep the ball in play on his or her side. As long as you keep hitting the ball back, the game goes on. When a player misses the ball, the ball hits the back wall behind that player's paddle and that player loses a life. When three lives are lost, the game is over and the other player wins.

## Code Overview

The application starts in Ping.java; this is where the main() method is that starts everything. This is also where the main game loop (run()) and main render function (render()) reside; these functions are very similar to what we described in Part 1 of this article.

Some of the other classes worth calling out here include:

- ***DynamicGameObjects:*** This class keeps track of all movable objects in the game, which includes the ball and the left and right paddles. In general, this class forwards actions on dynamic objects to all of the applicable objects. For example, when DynamicGameObjects.render() is called, that function simply calls render() on the ball and paddles.
- ***ForegroundObjects:*** This class keeps track of all the objects that will be painted on top of the game, such as the score. See Hud, below, for more details.
- ***StaticGameObjects:*** This class manages the placement and rendering of the game boundaries (the walls).
- ***Paddle:*** This class is responsible for rendering each paddle (the position is dependent on whether the paddle was instantiated as the right or left paddle) as well as tracking events. The Paddle instances listen for keyboard events and accumulate the events as described in Part 1 of this article. Then later Dynamic-GameObjects.gatherInput() is called, which defers to Paddle.gatherInput() to actually process the accumulated input events. The input events are processed to determine total down time as well as acceleration and, finally, paddle position for this frame.
- ***Ball:*** Holds the position and trajectory information for the ball and is responsible for updating that data through the updatePosition() method. This class also handles the collision logic for the ball in processCollisions().
- ***Collider:*** Handles the collision detection algorithm. It uses a simple parametric collision approach, where it detects the time at which a collision occurs for both the walls and the paddles.
- ***Background:*** Simply holds a background image for the game screen and handles rendering that image to the back buffer for every frame. Note the use of an intermediate image (bgImageScaled) to cache the scaled version of the image so that we need only copy the image and not scale it on the fly.
- ***GameLogic:*** Handles all of the information and state for making the game playable instead of just having a ball bouncing around on the screen.
- ***Hud:*** Handles the state and rendering of the "Heads Up Display," or the GUI that sits on top of the game. This includes things like the score and the help pane.
- ***PingFrame:*** This class extends JFrame and customizes our window for the game, including switching into fullscreen mode, setting the display mode, and handling keyboard events that affect the overall game.
- ***SoundFactory:*** Handles much of the simple audio processing, such as loading, starting, and stopping sound clips.
- ***ZoomyBackground***: This part of Ping is completely gratuitous, but we thought it looked cool. It's based on code that Jim Graham (another member of the Java 2D team) developed for a previous JavaOne conference to show some of the simple yet cool effects you can achieve with Java2D. We use this class as a mindless intermission animation while waiting for the user to start the game.

That should do it for the explanation of the program; we suggest you dive in at this point to see how things work.

## Summary

This article outlined a generic game framework that, although light on details, covers the essentials for what most games would need to do. It described some of the issues that game developers face, in terms of both algorithms and complexity as well as Java-specific programming.

The framework and examples developed in this article are mostly suitable for simpler 2D games; the complexities and issues in 3D programming are very specific to those types of games and we did not get into those rather more involved issues here. Note, however, that the main framework and rendering issues covered here are also issues in the 3D space, so anything gleaned here can only help in the larger world of 3D development.

Since there is so much that we didn't cover (and didn't have any hope of covering; game development is a hugely complex area of programming), it's worthwhile to pursue some of the resources we have listed below, as well as many other good sources of information in books and Web sites.

## Resources

- *The Ping source code and related information:* http://ping.dev.java.net
- *Information and forums specific to Java desktop client development:* http://javadesktop.org
- *Information and forums specific to Java game development. Projects include sample games as well as core game development technologies such as the Java OpenGL bindings:* http://javagaming.org
- *Information, forums, and projects for all Java developers:* http://java.net
- *Information about the overall Java platform, including articles such as the "Intermediate Images" mentioned above:* http://java.sun.com

JAVA DEVELOPER'S JOURNAL Advertiser **Index**

| Advertiser | URL | Phone | Page |
|---|---|---|---|
| Altova | www.altova.com | 978-816-1600 | 17 |
| Apple | www.apple.com/xserve | 877-41APPLE | 4-5 |
| Borland | www.go.borland.com/j1 | 831-431-1000 | 11 |
| Business Objects | www.businessobjects.com/dev/p7 | 888-333-6007 | 31 |
| Canoo Engineering AG | www.canoo.com/ulc | 888-333-6007 | 33 |
| Common Controls | www.common-controls.com | +49 (0) 6151/13 6 31-0 | 53 |
| DataDirect | www.datadirect.com/jdj | 800-876-3101 | 9 |
| Dice | www.dice.com | 877-386-3323 | 49 |
| EV1 Servers | www.ev1servers.net | 800-504-SURF | 21 |
| GlueCode Software | www.gluecode.com | 310-536-8355 | 39 |
| Google | www.google.com/cacm | 650-623-4000 | 43 |
| H&W Computer Systems | www.hwcs.com/jdj05.asp | 800-338-6692 | 45 |
| Information Storage & Security Journal | www.issjournal.com | 888-303-5282 | 57 |
| InstallShield | www.installshield.com/jdj | 800-809-5659 | 27 |
| InterSystems | www.intersystems.com/match1 | 617-621-0600 | 23 |
| M7 | www.m7.com/d7.do | 866-770-9770 | 35 |
| Microsoft | www.msdn.microsoft.com/visual/think | | 15 |
| Mindreef | www.mindreef.com | +1 603 465-2204 | 51 |
| Northwoods Software Corp. | www.nwoods.com/go | 800-434-9820 | 55 |
| Oak Grove Systems | www.oakgrovesystems.com | 818-440-1234 | 41 |
| Oracle | www.oracle.com/tools | 800-633-0759 | Cover II |
| Parasoft Corporation | www.parasoft.com/achievequality | 888-305-0041 | 29 |
| Qpass | www.qpass.com/jdj | 206-447-6000 | 47 |
| Quest Software, Inc. | www.quest.com/jdj | 800-663-4723 | Cover IV |
| SAP | www.sdn.sap.com | | 19 |
| Software FX | www.softwarefx.com | 800-392-4278 | Cover III |
| Stylus Studio | www.stylusstudio.com | 781-280-4488 | 6 |
| Web Services Edge 2005 East | www.sys-con.com/edge | 201-802-3069 | 60-61 |
| WebRenderer | www.webrenderer.com | +61 3 6226 6274 | 37 |

# Taking the Pain out of Large-Scale Java Development Projects

## Managing development environments

by Wolf Hengevoss
and Christopher Hearn

You know how to write good Java code and deployment to a server is no mystery either. But have you ever had to work in large development teams, maybe geographically dispersed (off-shoring…)? Ever had to address the pain of application software updates?

So often, when evaluating Java development tools, the focus is on productivity in writing code. While this is clearly important, it is essential not to underestimate the importance of managing the entire life cycle of an application – from setting up the development environment, to managing software delivery and maintaining the applications over the life cycle. The larger the project, the more critical this is, both from an efficiency and a financial point of view.

Over the past 30 years SAP has gathered in-depth experience in developing applications with large, geographically dispersed development teams. The delivery of software to the customer and the management of software upgrades and synchronizing these updates with customized code at the customer site are all issues that SAP has mastered. Now SAP is bringing the same high-quality approach to the Java world.

**Wolf Hengevoss** is a member of the SAP NetWeaver product management team.

*wolf.hengevoss@sap.com*

**Chris Hearn** is the product marketing director, SAP NetWeaver. Educated at Oxford University, England, Chris has 20 years of experience in IT in a variety of roles.

*christopher.hearn@sap.com*

## SAP Web Application Server

SAP Web Application Server is a key component of SAP's integration and application platform. SAP NetWeaver provides tool support for productive, model-driven and service-oriented Java development, but also, and this is key, an infrastructure that provides full-scale support for all phases of the life cycle – the Java Development Infrastructure (JDI). This infrastructure – tightly integrated with the SAP NetWeaver Developer Studio, delivered with the application server – reduces overall development TCO and provides for reliability and flexibility in the deployment and change management process.

### *Synchronized Team Development Made Easy*

The JDI gives you the best of both worlds – you can still develop in Java in your local environment, but your team's work is also synchronized via a central development environment.

Within the JDI, the Design Time Repository (DTR) handles file versioning to ensure that all developers are working from the same set of code. Developers access the central service via the SAP NetWeaver Developer Studio, check out files, produce new versions in the local file system, and check them back in after successful local testing. In the DTR perspective, you can compare versions, check version or revision history, and so on. During development, you can synchronize repository and local file systems whenever you wish, or even interrupt the connection between the two and reconnect later.

From the DTR you can manage:

- Different versions of a development object in the same repository
- Multiple states of a software component (development and consolidation of several releases)
- Multiple users making modifications to the same development object (with conflict detection)

Each state of software component development is represented in one workspace. The information about the state of a workspace can be propagated to other workspaces so you can synchronize the work of development teams using various instances of the DTR.

The DTR provides features for change management in a distributed, multiuser development environment. As you replace older versions of files with newer ones, the DTR handles this process centrally and keeps the version history. For more complex projects, though, you'll need more than that. Suppose modifications are occurring directly in end-users' systems, so various versions are being developed in parallel and multiple DTRs are in place. The version history is always deployed along with the files for global version history of the DTR. As a result, versions created in parallel are detected automatically across repository boundaries. Then there are times when, during code modifications, you may not want to have an earlier version automatically overwritten by updates – instead, the DTR supports the merging of two colliding versions, allowing you to combine the advantages of the newer version with your modifications. What's more, to reduce the maintenance effort as much as possible, you would want to integrate bug fixes from older releases into newer ones from the maintenance cycle. The DTR supports this, since changes are always deployed as a whole set of versions, instead of individually. This approach to change management means that the results are unaffected by the sequence in which the changes are applied. With its innovative approach to distributed development, the DTR enhances productivity and reduces costs of development throughout the whole product life cycle.

## Component-Based Development for Efficient Projects

The JDI takes a component-based approach to development. A component hierarchy distinguishes multiple levels of granularity:

- ***Development objects – the finest level of granularity:*** Tables, Java classes, and project files that are stored as versioned files.
- ***Development components (DCs):*** The units of the development and build process – they group development objects into larger development components, whereby one DC can contain multiple DCs.
- ***Software components:*** The deployment and installation units that are made up of DCs represent larger building blocks of products.
- ***Product:*** The complete solution.

DCs are at the heart of application development and follow a simple principle: only the parts declared to be public are visible to other DCs. In other words, to use one development component from another DC, you need to explicitly declare this usage. This explicit declaration of dependencies between DCs and precise relationships between objects allows the encapsulation of functionality that leads to a fine-tuned, highly efficient build process in the Component Build Service.

*Component Build Service*

Speed up the build process and avoid errors with the Component Build Service. Ever experience night-builds of complete applications that fail – followed by another 24-hour wait for the next build? With the JDI, these days are over.

Like the DTR, the Component Build Service (CBS) is a J2EE application that uses a database. It hosts all Java archives needed or produced during software development. For each software component state, a buildspace is set up to contain these archives. You can trigger a central build in the CBS at any time. Central builds apply to modified DCs only, along with any DCs that have dependencies with the changed archives.

This DC build approach allows you to correct errors in smaller chunks, reducing bug-fix cycle times. A failed build process will not affect the build process of any other DC.

The CBS also provides:

- J2EE cluster support for high performance
- Automated build scripts for Java development
- Automatic rebuild of dependent development components after changes to objects

After a successful build, the CBS automatically makes the sources and archives available for use by other developers. Because all archives are centrally stored and up-to-date in the CBS, it is an ideal source for retrieving or updating used libraries in your local file system. Faster build cycles and a current build environment significantly reduce development costs, time, and errors, especially in large projects.

## Managing the Development Environment

The development process starts with the definition of a product in the Software Landscape Directory. Here you define which software components are used in the product and the dependencies between them.

This information is imported into the Change Management Service (CMS) where you define the environment for the developers. You create workspaces and buildspaces by defining a track that's used to describe the development environment for one software component state. Fill the buildspaces with all the libraries required for a development configuration. This in turn is imported as an XML file into the SAP NetWeaver Developer Studio, defining the access to those objects the developer needs for his/her task.

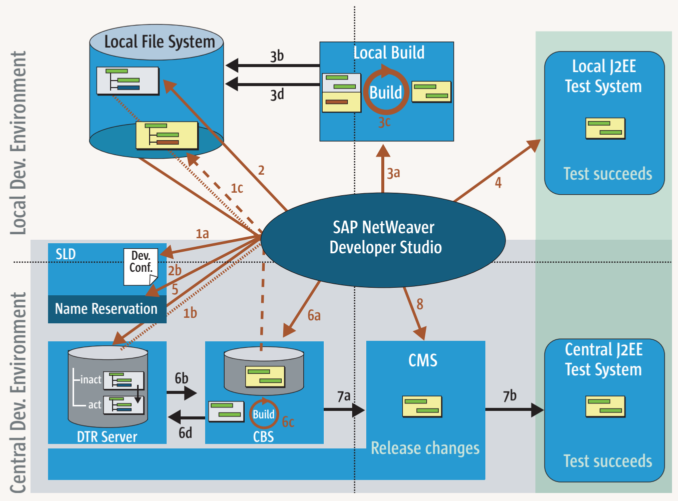
After development, the developer releases his/her work to the CMS again. Here the QM triggers the import into the consolidation system, after central releases approves and assembles a software component release. For the next release, it's not necessary to physically set up a new system; simply define a new track.

This gives you complete control of the product life cycle from product definition to application patches.

### Java Development with the JDI in Place

Figure 1 outlines the basic steps involved in synchronizing local and central file systems and then, after development, using the JDI's central build and deployment features.

1. Once the development configuration file is imported into the SAP NetWeaver Developer Studio (1a), you synchronize your local file system with the sources in the Design Time Repository (1b), and with the archives in the Component Build Service (1c). Then develop your Java application as you normally would on any local development environment.
2. Edit sources (check out, change, create, or add files).
3. Build archives locally.
4. Test your build results in a local test environment.



From here, the JDI automatically takes care of synchronization and even deployment into the test environment.

5. After successful testing, update (check in) sources in the DTR.
6. Build archives centrally. First, trigger the build (6a).The sources and archives are loaded automatically into the CBS (6b), then the build starts according to the provided build scripts (6c). After a successful build, sources are activated automatically( 6d).
7. Deployment to the central test environment is automatic.
8. Release changes for further processing.

## Key Takeaways

Let's summarize the advantages of this approach:

- The development infrastructure is set up and managed centrally.
- All sources are stored centrally in the DTR and retrieved for the central build process, and all archives are stored in the CBS, ensuring that you use the latest version of sources. You're assured a greater degree of safety when using archives.
- Clearly defining dependencies and encapsulating functions facilitates reuse and maintenance of development components.
- Distributed versioning and concurrency control allow you to manage large development projects taking place in different locations. Due to the DTR's versioning capabilities, modifications are not overwritten during updates, but can be integrated into the new version.
- You'll find a centrally managed system landscape – there's no need for each developer to know precisely where to deploy an archive.

For more information on the SAP Web Application Server and the Java Development Infrastructure or to download a trial copy check out www.sdn.sap.com.

web services EDGE conference & expo

# Web Services Edge 2005 East

**International Web Services Conference & Expo**

*New for 2005:*

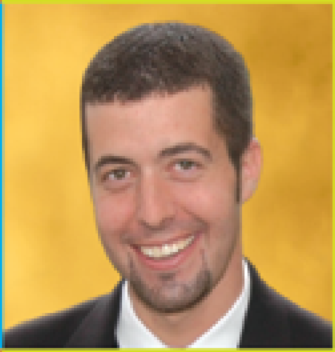
- **Colocating with LinuxWorld Conference & Expo** badges give access to both shows
- **Guaranteed Minimum Attendance 3000 Delegates**
- **New, Hot Sessions & Seminars**

## The Largest *i*-Technology Event of the Year!

**Tuesday, February 15**
**11 a.m.**

**Matt Ackley**
Senior Director, eBay Developers Program
**Topic:**
Web Services for eCommerce

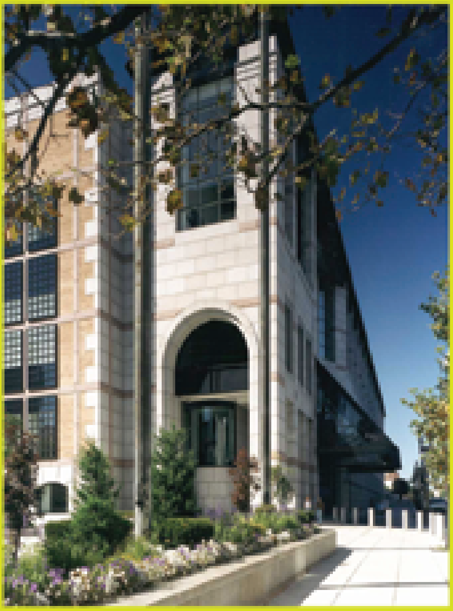
**Keynote Speakers & Featured Guests**



**Wednesday, February 16**
**11 a.m.**

**Ari Bixhorn**
Director, Web Services Strategies, Microsoft Corporation
**Topic:**
Indigo and the future of Web Services

**Anne Thomas Manes**
Burton Group
**Application Server Shoot-Out Facilitator**



**Hynes Convention Center**
**Boston, MA**
**February 15-17, 2005**

For Exhibit and Sponsorship Information:
Jim Hanchrow, 201-802-3066, jimh@sys-con.com

## Register Now! Hot Sessions & Seminars

- Ensuring Web Services Interoperability
- Web Services Standards: Going Behind the Mask
- The Role of Policy in Web Services Integration – It's More Than Just Security
- Four Abilities SOA will lack without a Registry
- Driving SOA Governance
- BPEL Best Practices from Real-World Projects
- SOA: From Pattern to Production
- Lessons From the Front Line – Building Interoperable Web Services
- Developing E-Commerce Applications with Web Services
- Developing Enterprise Class Web Services
- Orchestrating FORCEnet Engagement Packs with BPEL for Web Services
- CPI: A Global Integrated Problem Tracking and Resolution System using Java Web Services
- Using Service-Oriented Architecture and Web Services to Issue Business Licenses in the District of Columbia
- Developing Web Services with Eclipse
- The Interoperability Challenge of Web Services Security Standards
- Securing Web Services with WS-Security
- Anatomy of a Web Services Attack
- Using a Mobile Phone as an SSO Authentication Device in SOA Solutions
- XML Content Attacks
- XACML and Agnostic Authorities

Register Today: www.sys-con/edge2005east/registernew.cfm

*Come and see what everyone is talking about…*

**The Application Server Shootout!**

***Find out which server best fits your performance and ROI metrics***

## Join us in delivering the latest, freshest, and most proven Web services solutions…

At the *Fifth Annual Web Services Edge 2005 East – International Conference & Expo* as we bring together IT professionals, developers, policy makers, industry leaders and academics to share information and exchange ideas on technology trends and best practices in secure Web services and related topics.

## Exhibit in *The Web Services Pavilion* and see thousands of buyers!

Becoming a Web Services Edge Exhibitor, Sponsor or Partner offers you a unique opportunity to present your organization's message to a targeted audience of Web services professionals. Make your plans now to reach the most qualified software developers, engineers, system architects, analysts, consultants, group leaders, and C-level managers responsible for Web services, initiatives, deployment, development and management at the region's best-known IT business address– The Hynes Convention Center in Boston.

## 3-Day Conference & Education Program features:

- Daily keynotes from companies building successful and secure Web services
- Daily keynote panels from each technology track
- Over 60 sessions and seminars to choose from
- Daily training programs that will cover Web Service Security, J2EE, and ASP.NET
- FREE full-day tutorials on .NET, J2EE, MX, and WebSphere
- Opening night reception

PRODUCED BY SYS-CON EVENTS

**www.sys-con.com/edge**

Sponsored by: Information Storage+Security Journal, WLDJ, WebSphere Journal, JDJ, XML Journal, WebServices Journal, MX Developer's Journal, ea Industry Consortium, SD Times, .NET Developer's Journal, asp.netPRO

All brand and product names mentioned above are trade names, service marks or trademarks of their respective companies.

# From Within the Java Community Process Program

## *Moving the community forward*

Onno Kluyt

Before commercial developers choose a technology, they primarily ask two questions: How well does it solve a problem and how well does it provide a foundation to sell a solution to a problem. IT managers (and in-house developers) do their technology assessment homework a bit differently: they look first at how well the technology solves a problem and the longevity of a solution to a problem. The common denominator is the desire to feel "technology safe." Aspects that contribute to the sense of safety are whether the technology is evolving, whether its ecology is expanding, whether there is a choice in vendors, and whether there is equal opportunity.

In our industry – as in many other places – there is a circle of adoption: developers create programs, programs run on platforms, platforms represent volume, volume attracts developers who create programs, programs attract end users, end users create volume, and so forth. In such circles there are two spots where key choices are made, must be made:

- Differences between platforms cause developers to either duplicate their software creation or software validation efforts. This causes a fragmentation of the developer's economic opportunity.
- Differences between similar platforms decrease vendor choice for the end user, which increases the risks for vendor lock-in causing fragmentation of the end-user's economic opportunity.

Is this an argument for complete commonality between platforms? No, that is both unachievable and undesirable. There would be no reason to invest in building today's and tomorrow's complex systems if there were no means by which vendors can differentiate. The skill of the game is to steer the froth ahead of the standard. New techniques will be tried in the market and enjoy differing levels of success. Some of these techniques will flow into the standards process. The standards process must navigate through vendors' temptations to obtain competitive advantage via the contents of a new specification and end-users' needs for a commodity standard. Sometimes this balance has been captured, perhaps crudely so, in the expression "collaborate on specifications, compete on implementations" – collaboration, with and within the community. Collaboration that has as its measure of success that it moves the community forward. The community has not succeeded when the first person can claim compliance, but when all in the community can compete and conform.

This equal opportunity for the whole community necessitates that end users have a means to rate the claims the vendors are making, and that vendors are capable of knowing what effort sits behind a claim of compliance. This equal opportunity makes it obvious that the specification must be the standard of compliance, not any one particular implementation. This leads to three fundamental requirements: the specification is of high quality and unambiguous, it is well known, and an implementation's conformance can be measured objectively. This is why the Java Community mandates the creation of a reference implementation and compatibility test suite in addition to just a specification. Not because it's fun to do so nor easy, but because these are necessary tools for implementers to be free to compete on known terms, for end users to be free from vendor lock-in, and for developers to have the confidence that the Java programs they create are not lied to by the implementations they need to run on. That is what positively feeds the circle of adoption, leading to an expanding ecology for all of us to enjoy.

We do compatibility not because it is easy but because it's worthwhile. I admit this comes with some fundamentals we must all agree to but it is worthwhile to do so. The presence of a reference implementation ensures that real engineers have actually succeeded in implementing a specification before that specification is declared final, is declared a standard. In addition, many previous conformance attempts at other standard-setting organizations proved that you cannot build a meaningful test suite without an implementation to test it against. Both provide a positive feedback loop on the quality and correctness of the specification, since in their development both will be interpreting the specification while the specification is being developed. A reference implementation must be well known and readily available. The test suite will not be able to test more than what the reference implementation does and so a vendor will want to know (and have access to) the implementation that may have been built by its competitor that is leading the specific standardization effort.

The result is the emergence of multiple implementations from many sources based on a specification that developers using the Java technology can rely on, knowing that the functionality guaranteed by the specification is indeed present in the platform of their vendor of choice. Hence, the ability to feel safe. Together we can make this result happen and enjoy the opportunities it provides by participating in the community and with the community.

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

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

*onno@jcp.org*

ENTERPRISE CHARTING SOLUTIONS

Need More Than A Pretty Face?

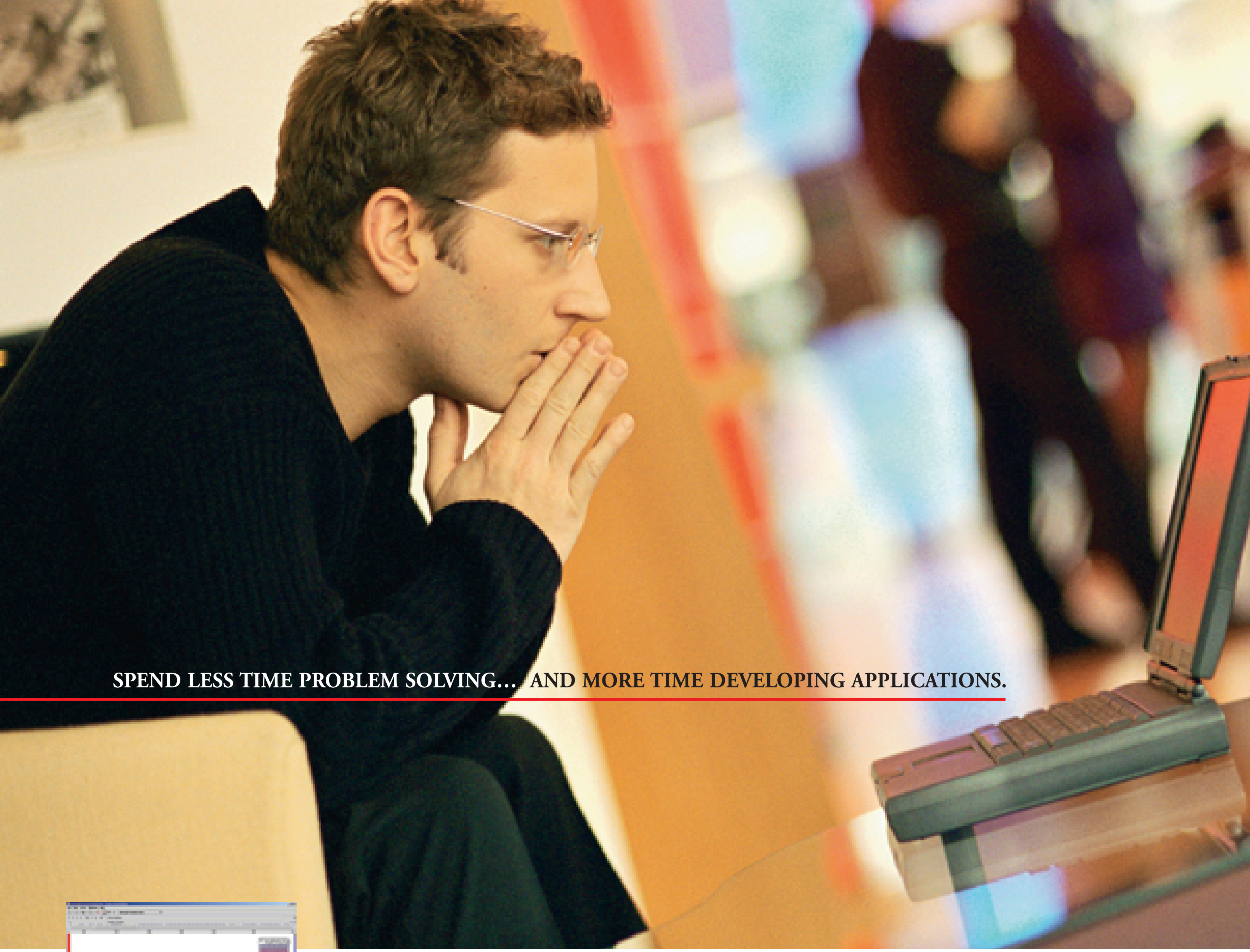
For Serious Enterprise-Level
Data Visualization & Analysis

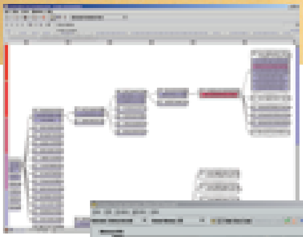
Multi-Platform
Ease of Use
Outstanding Support
...and A Pretty Face!

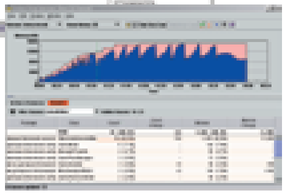
Extensibility
Performance
Scalability

Chart FX

www.softwarefx.com


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


PerformaSure – a system-wide performance diagnostic tool for multi-tiered J2EE applications running in test or production environments.


JProbe – a performance tuning toolkit for Java developers.

**Join The Thousands of Companies Improving Java Application Performance with Quest Software.**

Whether it's a memory leak or other performance issues, Quest Software's award-winning Java products — including JProbe® and PerformaSure™ — help you spend less time trouble-shooting and more time on the things that matter. Quest's Java tools will identify and diagnose a problem all the way down to the line of code, so you no longer have to waste time pointing fingers or guessing where the problem lies. Maximize your team's productivity with Quest Software by downloading a free eval today from **http://www.quest.com/jdj**.

QUEST SOFTWARE®