SYS-CON

JAVA DEVELOPER'S JOURNAL EDITORS' CHOICE AWARDS PAGE 54

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

JDJ

JDJ.SYS-CON.COM VOL.11 ISSUE:8

AjaxWorld Conference & Expo OCT 3-4 2006

SEE PAGES 54-55 FOR DETAILS

SANTA CLARA CONVENTION CENTER
WWW.AJAXWORLDEXPO.COM

PLUS...

- Jakarta Struts & JavaServer Faces

Building Real-Time Applications with Continuous Query Technology

RETAILERS PLEASE DISPLAY UNTIL OCTOBER 31, 2006

$5.99US $6.99CAN

09

0 09281 01751 6

Fireproof Your Code

Prevent Java code fires with JProbe®

Constantly fighting Java code fires? Prevent infernos — before code moves into production — with award-winning JProbe from Quest Software.

Quickly pinpoint Java code hot spots with line-level analysis. Discover and debug memory leaks to dramatically improve performance. Automate the task of performing code analysis during off-peak hours. And release applications with confidence, knowing they have been fully tested. JProbe is the proactive solution that gives you higher levels of productivity and end user satisfaction.

Stop Java code performance flare ups — before they start. Improve code quality and increase application efficiency with JProbe.

Watch JProbe in action. View the new product demo at:
www.quest.com/JavaCode

Application Management | Database Management | Windows Management

QUEST SOFTWARE®

Unofficial History of Programming: '96 – '06

By Yakov Fain

I want back in the '90s...seriously. Ten years ago I didn't know Java: I'd been using PowerBuilder and was able to program pretty much everything in this RAD object-oriented tool. To find a job back then, all I needed to have on my résumé was PB, a single framework (PFC), and SQL. With these skills I could have created a prototype of a rich CRUD client/server application in a couple of days. However, that was the sunset of the client/server era.

While making the deployment of the client software easier, the Web pushed the user-facing applications years back. Just look at these ugly screens: several plain text boxes, a dropdown, and a trivial HTML table. Mainframe dumb terminals had black screens with green letters, but the interaction with the big iron was super fast. The Web offered a white background with black letters and poor performance. But the entire world was so happy with this new way of accessing the wealth of data and tons of e-commerce opportunities, that people were willing to put up with some minor inconveniences.

GoF had released a famous book on design patterns. I wonder if anyone has read this manuscript from start to finish? This book was the first step in turning programming from an art to a trade. Singleton, MVC, Factories, value objects…just pick up the proper design pattern(s), and your code will look as if it was written by an expert. Don't forget to comment your programs explaining which design patterns were used in your code. There still is a small number of programmers who get by without pattern programming, but they'll be extinct soon.

SQL was in favor in the '90s. People knew how to delete duplicates from a database table by applying such SQL clauses as group by and having. How many people have read the book by Joe Celko, SQL for Smarties? Let me put it another way. How many people know what SQL is? Why bother, Hibernate will let me map class attributes to the database table columns. How nice…I'm drowning in XML now. Let's not jump ahead though; mankind did not know Hibernate or XML back then.

The Java programming language was born. It became visible as a language for creating applets, but it quickly abandoned the desktop and started to shine on the server side. It took Sun almost 10 years to realize that desktop programming is also important and it's time to create a Swing-based RAD tool.

The end of the last century can be called the Gold Rush of Programming. People started to spread the fear of Y2K issues. Since the dates were stored as two digits, some nuclear explosion or a less serious disaster was expected on January 1, 2000. For example, I'd never include "'96–'06" in the title of this article. Why? Because 06 minus 96 is equal to negative 10. Get it? Lots of people quickly became programmers with the noble mission of saving mankind. Lots of IT managers quickly climbed the corporate ladder working on this noble mission.

In the beginning of the new century, XML became popular. Yes, it was a nice way to describe data, but at the same time it was too heavy. It did not manage to kill the CSV format – the hype is over – but it did find its use in a variety of applications.

Microsoft came out with .NET platform, which became a direct competitor of J2EE. These two mainstream technologies cover most of the enterprise software development.

Another important trend of this century is the spread of open source software. In the past, vendors used to sell software licenses, but now many of them give the software away for free and sell services instead. The documentation of our open source product may be poor, but no worries, we'll be happy to help you with our great tool for an extra fee.

What are the latest notable trends? Let me throw in a couple of buzzwords.

– continued on page 59

JAVA DEVELOPER'S JOURNAL

Editorial Board

Java EE Editor: Yakov Fain
Desktop Java Editor: Joe Winchester
Eclipse Editor: Bill Dudney
Enterprise Editor: Ajit Sagar
Java ME Editor: Michael Yuan
Back Page Editor: Jason Bell
Contributing Editor: Calvin Austin
Contributing Editor: Rick Hightower
Contributing Editor: Tilak Mitra
Founding Editor: Sean Rhody

Production

Associate Art Director: Tami Lima
Executive Editor: Nancy Valentine
Research Editor: Bahadir Karuv, PhD

To submit a proposal for an article, go to http://jdj.sys-con.com/main/proposal.htm

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.

ALTOVA®
umodel®
2006

Visualize works of software art

Draw on UModel® 2006, and picture better programs based on UML™.

Spied in UModel 2006 Release 2:

- Activity diagrams
- State machine diagrams
- Component structure diagrams
- Context-specific toolbars

Altova UModel 2006, the burgeoning new force in the software design space, is the simple, cost-effective way to draw on UML. Use it to interpret or create your software architecture. Decode Java or C# programs into clear, accurate UML2 diagrams, or outline applications and generate code from your plans. With all major diagram types, interoperability via XMI 2.1, and an artful user interface, UModel makes visual software design practical for programmers and project managers everywhere. Take the mystery out of UML!

Download UModel® 2006 today:
www.altova.com

ALTOVA®

AUGUST 2006 VOLUME:11 ISSUE:8

JDJ contents

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

Where Are the High-Level Open Source Design Tools?

by Ed Merks

In answer to the question "Where are the high-level Open Source design tools for Java?" I believe that they're emerging from efforts at Eclipse.org. These efforts began with the Eclipse Modeling Framework (EMF) in 2002, and have been building momentum ever since, with the addition of the UML2 project, the Graphical Modeling Framework (GMF), the Generative Modeling Tools (GMT) Project, and Model Driven Data Integration (MDDI). More recently, with the creation of the new top-level Eclipse Modeling Project (http://www.eclipse.org/modeling) to act as a home and focal point for all of these modeling related technologies, there is clearly an ever-growing focus in this area. So what does it all mean and what's behind the acronyms?

The focus of the Modeling Project is on bridging the design-versus-development gap using a bottom-up approach. We're building practical development tools and frameworks, evolving them over time, and using them ourselves to build the subsequent layers of the onion.

For example, given just an XML Schema as input, EMF and GMF can generate a fully functional graphical application ready to be tailored and specialized. Furthermore, since the generator technology supports merging, a developer can switch between modeling and hand-coding. This is essential for the adoption of high-level tools by a community that likes to view them as a dimmer switch it can adjust to suit various skill levels, rather than "all dark" or "all light" tools that replace hand-coding. And since the model can be specified in the form of annotated Java (which the generator produces to support round-tripping) it's possible for a developer to work completely in Java without buying into the whole sales pitch of the model-driven approach.

We validate these ideas by eating our own dog food: EMF models are used extensively throughout the Modeling Project, and many of our tools are based on EMF-generated editors. Now, with GMF maturing, we're beginning to use it as the basis for more sophisticated, user-friendly tools.

The Eclipse modeling tools are trying to appeal to as broad a target audience as possible by supporting many different development styles, something that hasn't occurred in the past and that, hopefully, will dispel the myth that modeling and coding are mutually exclusive or just rigid one-way processes.

I believe the main stumbling blocks to acceptance of design tools have been the rigidity and complexity of the methodology itself and the poor quality of the artifacts that they generate. To address the former issue, the Eclipse modeling tools have focused on building exemplary support around a very simple core model rather than around something much more complex. To address the latter issue, we've focused on producing code of handwritten quality. If a tool doesn't produce high-quality code, language-fluent developers will often reject it.

To understand the target audience for modeling tools, I often relate back to my own experiences and how my thinking has evolved over the years. When I was first exposed to MOF (Meta Object Facility), I didn't know how to read UML diagrams, so I was immediately convinced that they were a useless diversion. "What's wrong with plain old Javadoc?" Then, I looked at the generated code, and I was immediately convinced that it was too verbose and inefficient to be of any use at runtime. "I'd never write such garbage by hand!" And, when forced to learn the meta-model itself, I was immediately convinced that it was full of extraneous baggage. "Who needs all this stuff?"

Despite my "well considered" objections, I continued to use MOF in my day job and had much time to reconsider. Having learned to read a class diagram, it didn't take long to be convinced of the cliché "a picture is worth a thousand words." It also became clear that if it's silly to draw a picture of a labeled box containing a labeled feature, it's even sillier to write by hand, possibly thousands of times, an interface with a getter, a setter, an implementation class with the same things as well as a field to store the data, and last but not least, a factory method to create the instance. All of this is menial work that's beneath the skilled developer. Ironically, the very simplicity of the diagrams that made them seem silly is precisely where their value lies.

The code being generated was of poor machine-written quality, but it quickly became clear that this could easily be addressed by producing simple, clean code that looked exactly like I'd write by hand. For example, a getter need not do anything more than return the value of a variable. And, by using a template-based approach, we could provide flexibility and control to any developer, letting him tailor what's produced by the generator to fit his specific needs or tastes. Add to this a generator that can merge its output with existing code, and you reach my tipping point: a

– continued on page 59

SYS-CON MEDIA

President and CEO:
Fuat Kircaali *fuat@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*
Robyn Forma *robyn@sys-con.com*
Advertising Sales Manager:
Megan Mussa *megan@sys-con.com*
Associate Sales Managers:
Kerry Mealia *kerry@sys-con.com*
Lauren Orsi *lauren@sys-con.com*

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

Production
Lead Designer:
Louis F. Cuffari *louis@sys-con.com*
Art Director:
Alex Botero *alex@sys-con.com*
Associate Art Directors:
Abraham Addo *abraham@sys-con.com*
Tami Lima *tami@sys-con.com*

Web Services
Information Systems Consultant:
Robert Diamond *robert@sys-con.com*
Web Designers:
Stephen Kilmurray *stephen@sys-con.com*
Wayne Uffleman *wayne@sys-con.com*

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

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

_INFRASTRUCTURE LOG

_DAY 15: This project is out of control. The development team's trying to write apps supporting a service oriented architecture...but it's taking FOREVER!

_DAY 16: Gil has resorted to giving the team coffee IVs. Now they're on java while using JAVA. Oh, the irony.

_DAY 18: I've found a better way: IBM Rational. It's a modular software development platform based on Eclipse that helps the team model, assemble, deploy and manage SOA projects. The whole process is simpler, faster and all our apps are flexible and reusable. :)

_The team says it's nice to taste coffee again, but drinking it is sooo inefficient!

IBM®

Rational®

Download the IBM Software Architect Kit at:
IBM.COM/**TAKEBACKCONTROL**/FLEXIBLE

Integral Java: A Single Solution for Bypassing the Pitfalls of Split Stacks

A new single stack platform, one set of APIs for everything, and the future of mobile Java

by John McCready

Java, in the form of the Java 2 Platform Micro Edition (J2ME), has become a prerequisite for all future mobile handsets for at least the next seven to nine years. Not only will the core applications needed for the user experience be created in Java, it will also serve as the basis for the lucrative downloadable application market – the Java segment of which is currently projected to exceed $15 billion by 2008.

If there are any remaining questions as to the pivotal role Java is destined to play in the mobile industry, consider the following numbers. By mid-2006, the installed base of Java enabled handsets will cross the billion-unit mark, with over 35 vendors already offering upwards of 600 different Java-enabled handset models. Furthermore, over four million software developers, per 3G Americas' estimates in mid-2005, are now involved in creating J2ME-specific software to feed, as well as fuel, this burgeoning demand for Java-based functionality for mobile handsets.

Despite all of these inescapable positives, handset manufacturers are still facing a major challenge on the Java front. Their approach to providing Java capability on handsets has become passé – plagued by inefficiencies, fraught with application integration complications, and above all, vulnerable to security risks. All of these problems stem from the fact that manufacturers never set out to fully and tightly integrate the necessary Java platform [i.e. J2ME] with the native handset OS kernel, the handset engines/codecs, and the native libraries. With this two-stack approach, there exists a native stack with its own set of exposed APIs. Then, there's the Java stack, with exposed APIs also grafted atop the native stack.

In short, the two-stack approach offers a seriously flawed foundation unsuitable to support a Java future.

Yet, groundbreaking developments have been taking place and a much better way to implement mobile Java has emerged, one that permits handset makers to overcome all of the problems associated with the conventional two-stack approach. With this new platform, manufacturers can provide developers with a single-stack that supports all the strategic Java APIs, while providing the necessary access to the native engines, codecs, and libraries via the Java APIs. It's a real win-win solution without any limitations or drawbacks.

The Conventional Two-Stack Approach to Java

In 2001, handset makers were confronted with the need to support Java, primarily to accommodate downloadable Java applications. At that juncture, the native handset OS (Symbian), the OS services, and middleware (access to the handset engines and codecs), the handset services (phone settings), handset application (Web browser), and the native libraries were all being developed in C. Rather than making any changes to this native infrastructure to tightly integrate Java, manufacturers opted instead to create a host-porting layer on top of their native libraries as the basis of their Java support.

A Java KVM was then implemented on top of the host-porting layer, where a KVM is a J2ME-specific subset of a Java virtual machine (JVM). This KVM then served as the platform for Java libraries, with their Java APIs. Suffice it to say, this was not a very efficient way to realize Java. The two-stack approach compromises Java performance, compounds application integration complexity, and at a minimum doubles the amount of software testing (and quality assurance) that has to be performed. All of that, bad enough as it is, is not the limit of the pitfalls associated with this approach.

This two-stack approach, with native APIs exposed to all, is also an unmitigated security nightmare. Access to the native APIs, as manufacturers and operators are acutely aware of, enables hackers to easily create malicious viruses, worms, and spyware. Thus, the current goal, across the industry, is to try to restrict access to the native APIs. However, with the two-stack approach it's difficult to restrict access to the native APIs since bona fide developers have to use these APIs to interact with libraries, codecs, and engines.

Fortunately, a new single-stack approach provides an elegant and universal solution to this security problem as well as all the other complexity and inefficiency-related issues associated with two stacks.

The Single-Stack Java Solution

With a single-stack platform, handset makers no longer have to expose their native APIs to the software development community-at-large. Software developers can instead use one set of strategic broad-spectrum Java APIs for all of their needs includ-

John McCready is senior vice-president of marketing for SavaJe Technologies, developers of the most advanced Java technology-based mobile operating platform. The SavaJe-based Jasper S20 mobile phone was named "Device of the Show" at the 2006 JavaOne Conference. The SavaJe Mobile Platform radically simplifies and accelerates the development of highly customizable, richly branded, and secure user interfaces across mobile feature phone handsets.

ing that of accessing the handset's native engines and codecs. The graphs below illustrate how the single-stack approach markedly differs from the conventional two-stack approach that manufacturers have been employing.

To ensure that software developers gain uncompromised access to everything they need on the handset using just Java APIs, the single-stack platform goes well beyond just implementing the basic amount of functionality required to be J2ME-compliant. Unlike Java 2 Enterprise Edition (J2EE) and Standard Edition (J2SE) – where there's only one version of the platform – J2ME (as shown in Figure 2) has two variants known as configurations. The Java functionality available with each of the two configurations is defined in terms of the core libraries to be included with that configuration as well as by the capabilities of the Java virtual machine associated with it.

The two different J2ME configurations are:

1. Connected Limited Device Configuration (CLDC), and
2. Connected Device Configuration (CDC).

CDLC, as noted by the "limited" in its name, is meant for low-cost, limited-function devices, while CDC is for more sophisticated mobile devices. CLDC, as shown in Figure 2, can be implemented with a KVM (alimited function subset of a JVM), while CDC, in common with J2EE and J2SE, requires a full JVM. Thus, CDC, from the get-go, has more in common with mainstream Java than CDLC does.

CLDC and CDC, as again shown in Figure 2, have so-called profiles implemented on top of the configurations. These profiles define the requisite Java software functionality and the API repertoire for a specific class of device. At present there are two key profiles defined for J2ME: the Mobile Information Device Profile (MIDP) and Personal Profile (PP). MIDP (now at version 2.0), which is to be used with CLDC, defines basic connectivity, persistent storage, networking, and user interface functionality. MIDP is targeted at low-end handsets. On the other hand, PP, meant to be used with CDC, is for high-end devices, including PDAs.

The single-stack platform implements both CDC as well as CLDC/MIDP2.0. It thus provides software developers with a complete set of J2ME APIs that is more comprehensive, feature-rich, and powerful than the APIs available with just MIDP. This is the crux of the solution. Thanks to the availability of this expanded set of Java APIs, software developers are no longer as dependent on native APIs as they previously were. Now they can use Java APIs for all needs, whether it's to develop core handset utilities or value-added, downloadable applications.

The advantages of this single-stack approach are many and obvious, positively benefiting not just the handset makers but also developers, operators, service providers, and even users. The key advantages of the single-stack approach include:

- Reduction in development and testing costs by reduced software duplication and complexity;
- Major improvement in software security, greatly minimizing the disruptive threats of viruses and spyware, by obviating the need for exposed native APIs;
- Simplifying the data sharing between applications (chat and address book) through the use of common APIs and libraries;
- Standardization on Java, without the need to flip-flop between Java and native code, thus promoting the creation of a more consistent and cohesive user experience;
- Reducing development and testing schedules, expediting overall time-to-market;
- Tangible increases in Java performance by tighter integration with the OS kernel (without using an intermediary host porting layer).

The Bottom Line

Java has become a mandatory prerequisite for future handsets. The conventional approach of implementing Java as a separate stack grafted on top of the native libraries via a host-porting layer is fraught with problems – security, performance, and inefficiency being key among these. The new single-stack platform, which with a stroke implements both J2ME CDC and CLDC/MIDP2.0, eliminates all of the problems associated with the two-stack approach. Rather than having to contend with multiple stacks, with two competing sets of APIs, the single-stack approach provides one unified stack with one set of APIs. By reducing software duplication and complexity, this new approach reduces costs, expedites time-to-market, enhances security, enforces consistency, and increases performance. It, in reality, is the only way to go forward when it comes to J2ME on mobile handsets. Period.

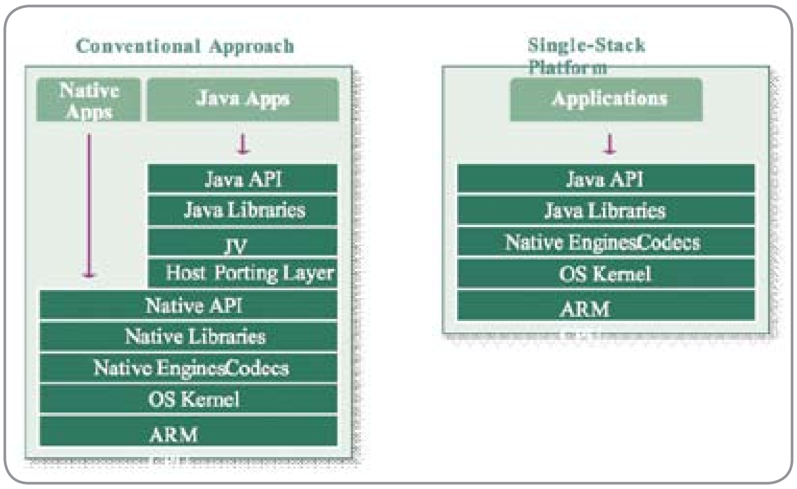


Figure 1 An all-encompassing single-stack approach eliminates, at a stroke, all the inefficiencies, complexities, duplication, and security concerns associated with the conventional two-stack approach.

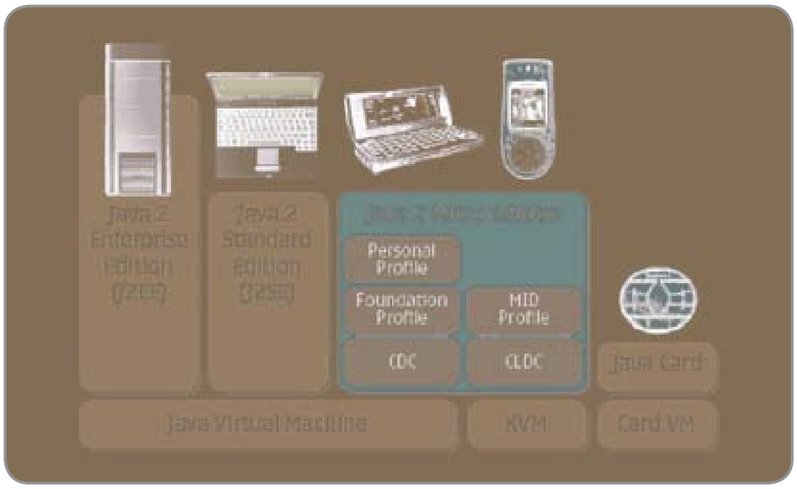


Figure 2 In contrast to J2EE and J2SE where there are only one version of the platform, J2ME comes in two variants known as configurations

Jakarta Struts & JavaServer Faces

The add-and-remove pattern

by Heman Robinson

A previous article compared Jakarta Struts and JavaServer Faces implementations of five simple design patterns for list selection. (JDJ, Vol. 11, Issue 3).

Long lists and ordered selections require a more complex design pattern. This pattern displays available items in one list and chosen items in another so the user's choices are always visible and easily modified.

This design pattern is commonly called a Dual List or Dual Listbox selector. It is also known as the Selection Summary or List Builder pattern. In the Java Look and Feel Design Guidelines, it's called the Add-and-Remove pattern:

Typical Struts implementations of this pattern require JSPs, Java, and JavaScript. JavaServer Faces doesn't need JSPs, but they're used here for easy comparison.

Listing 1 shows the JSP for Struts and Listing 2 shows the JSP for JavaServer Faces. Complete code listings for the backing beans, config files, and other code for all six list selection patterns can be downloaded from the JDJ Web site.

Jakarta Struts Implementation

Using Jakarta Struts, the JSP for this pattern defines a table layout with three columns. In the first and third columns, the Available and Chosen lists are implemented using the Struts tags <html:select> and <html:optionsCollection>. For internationalization, the labels can use the <fmt:message> tag from the JavaServer Pages Standard Tag Library. Many people find Struts and JSTL tags a powerful combination.

```
<table border="0" cellpadding="0" cellspacing="5">
   <tr align="middle" valign="center">
      <td>
         <fmt:message key="titles.available"/><br />
         <html:select property="availableValues"
            multiple="true" size="7"
            style="width:80px;" styleId="available"
            onchange="doUpdate( false, true );">
            <html:optionsCollection
               property="availableList"/>
         </html:select>
      </td>
         ...
      <td>
         <fmt:message key="titles.chosen"/><br />
         <html:select property="chosenValues"
            multiple="true" size="7"
            style="width:80px;" styleId="chosen"
            onchange="doUpdate( true, false );">
            <html:optionsCollection
               property="chosenList"/>
         </html:select>
      </td>
   </tr>
</table>
```

The form bean contains the "availableList," "availableValues," "chosenList," and "chosenValues" properties used in the JSP.

```
private List availableList = new ArrayList();
private String[] availableValues = new String[ 0 ];
private List chosenList = new ArrayList();
private String[] chosenValues = new String[ 0 ];
   ...
public List getAvailableList()
public void setAvailableList( List list )
public String[] getAvailableValues()
public void setAvailableValues( String[] list )
public List getChosenList()
public void setChosenList( List list )
public String[] getChosenValues()
public void setChosenValues( String[] list )
   ...
```

The "availableList" and "chosenList" properties store the lists of available values as LabelValueBeans. The "availableValues" and "chosenValues" properties store the selected values as arrays of Strings.

For the Add-and-Remove pattern, these selected values are of no interest. We don't have to tell the server which values are selected but which items appear in the list contents.

However, when forms are submitted, list contents don't get sent back to the server; only their selected values do. This is usually the most efficient way to process forms, but for this design pattern it's a problem.

There are several ways to solve this problem. One way is to submit the form every time the list contents change. This produces excess screen refreshes and network traffic. Another way is to use AJAX technology to communicate with the server. This reduces screen refreshes, but still generates network traffic. There's no need to generate network traffic until the user completes their changes.

The best solution is to store the selected values in a hidden control. This way the values get sent back to the server only once, when the form is submitted. It's sufficient to store only the chosen values; changes to both lists can be derived from them. In this example, we'll store the chosen values as a delimited string in a hidden text field.

Using this hidden field, the buttons in the middle column can be implemented as follows:

```
<td>
   <br />
   <input type="button" style="width:100px;"
      id="add" onclick="
      doMove( 'available', 'chosen', false );"
      value="<fmt:message key='add'/>"    /><br />
   <input type="button" style="width:100px;"
      id="addAll" onclick=
      "doMove( 'available', 'chosen',  true );"
      value="<fmt:message key='addAll'/>" /><br />
   <br />
   <input type="button" style="width:100px;"
      id="remove" onclick=
      "doMove( 'chosen', 'available', false );"
      value="<fmt:message key='remove'/>" /><br />
   <input type="button" style="width:100px;"
      id="removeAll" onclick=
      "doMove( 'chosen', 'available',  true );"
      value="<fmt:message key='removeAll'/>" />
   <html:hidden styleId="chosenItem"
      property="chosenItem" />
</td>
```

The buttons are implemented using standard HTML <input> tags. The <html:hidden> field stores the chosen items as a delimited string. The items are stored by invoking the JavaScript "doMove()" function, which performs all four of the button actions:

```
/**
 * Move selected items between lists.
 * <p>
 * @param  sourceId  ID of source list
 * @param  destId    ID of destination list
 * @param  all       true iff moving all
 */
function doMove( sourceId, destId, all )
{
   // Move the items between the lists.
   var sourceElem =
      document.getElementById( sourceId );
   var destElem =
      document.getElementById( destId );
   for( var i = 0; ( i < sourceElem.length ); )
   {  if( sourceElem.options[ i ].selected || all )
      {  var newOption =
            document.createElement( "OPTION" );
         newOption.text =
            sourceElem.options[ i ].text;
         newOption.value =
            sourceElem.options[ i ].value;
         destElem.options[ destElem.length ] =
            newOption;
         sourceElem.remove( i );
      }
      else
         i++;
   }

   // Update the button states.
   doUpdate( false, false );

   // Store the chosen items in a hidden field.
   var chosenItem =
      document.getElementById( "chosenItem" );
   var chosenList =
      document.getElementById( "chosen" );
   chosenItem.value = "";
   for( var i = 0; ( i < chosenList.length ); i++ )
   {  chosenItem.value +=
         chosenList.options[ i ].value + '|';
   }
}
```

Besides implementing the button actions, it eases the users' learning curve to disable these actions when they don't make sense. For example, when there are no selected items, the "Add" and "Remove" buttons can't be used. When the available list or chosen list is empty, the "Add All" or "Remove All" button can't be used.

The "doUpdate()" function enables or disables the buttons based on the user's selections and the list contents.

"JSF provides a natural migration path **for projects moving from Struts"**

The "doUpdate()" function is invoked from the "doMove()" function above as an "onchange" handler for the lists, and as an "onload" handler in the <body> tag to initialize the button states.

```
/**
 * Update the button states based on whether
 * lists have contents and selected items.
 * Deselect list items if requested to ensure
 * at most one list contains selections.
 * <p>
 * @param  offAvailable  deselect available list
 * @param  offChosen     deselect chosen list
 */
function doUpdate( offAvailable, offChosen )
{
   // Get the lists and deselect if requested.
   var availableList =
      document.getElementById( "available" );
   var chosenList =
      document.getElementById( "chosen" );
   if( offAvailable )
      availableList.selectedIndex = -1;
   if( offChosen )
      chosenList.selectedIndex = -1;
  // Update the button states.
    document.getElementById( "addAll" ).disabled =
       ( availableList.length == 0 );
    document.getElementById( "removeAll" ).disabled =
       ( chosenList.length == 0 );
    document.getElementById( "add" ).disabled =
       ( availableList.selectedIndex < 0 );
    document.getElementById( "remove" ).disabled =
       ( chosenList.selectedIndex < 0 );
}
```

Using these JavaScript functions, list contents are correctly updated and the user's chosen items are stored in the hidden field. When the form is submitted, the user's "chosen" list is read from the hidden field. In this example, it's used to re-populate both lists to reflect the user's choices.

Lists are re-populated by using the "languageList" in the form bean. This is a constant list containing all possible choices. The "move" method of the form bean manipulates the "available" and "chosen" lists based on the contents of the hidden "chosenItem" field.

```
private List languageList = new ArrayList();
private String chosenItem = "";
   ...
public List getLanguageList()
public void setLanguageList( List list )
```

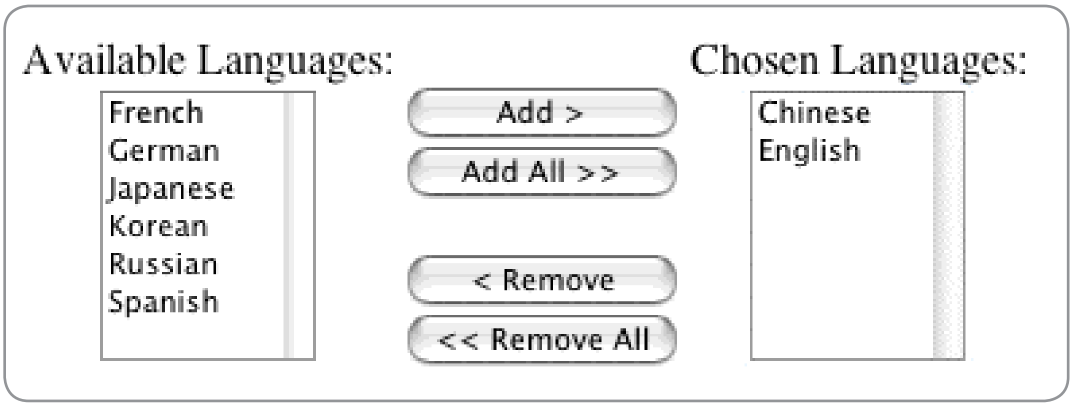


Figure 1 Add-and-remove pattern

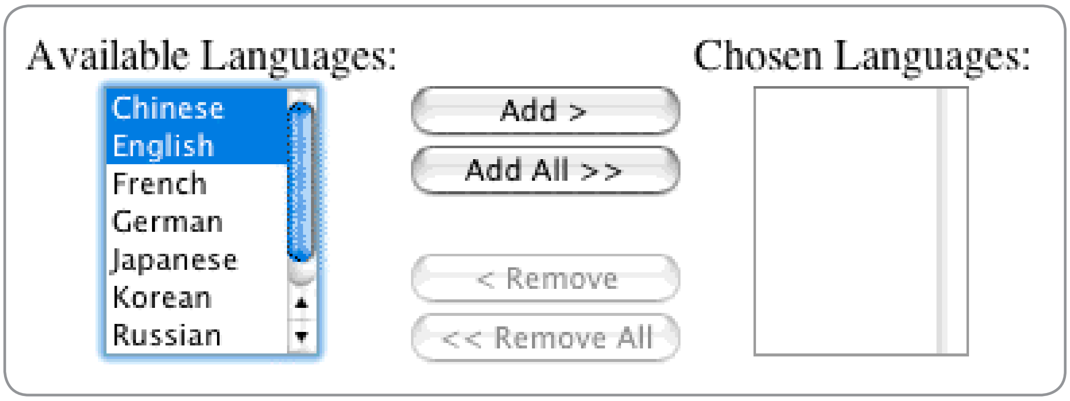
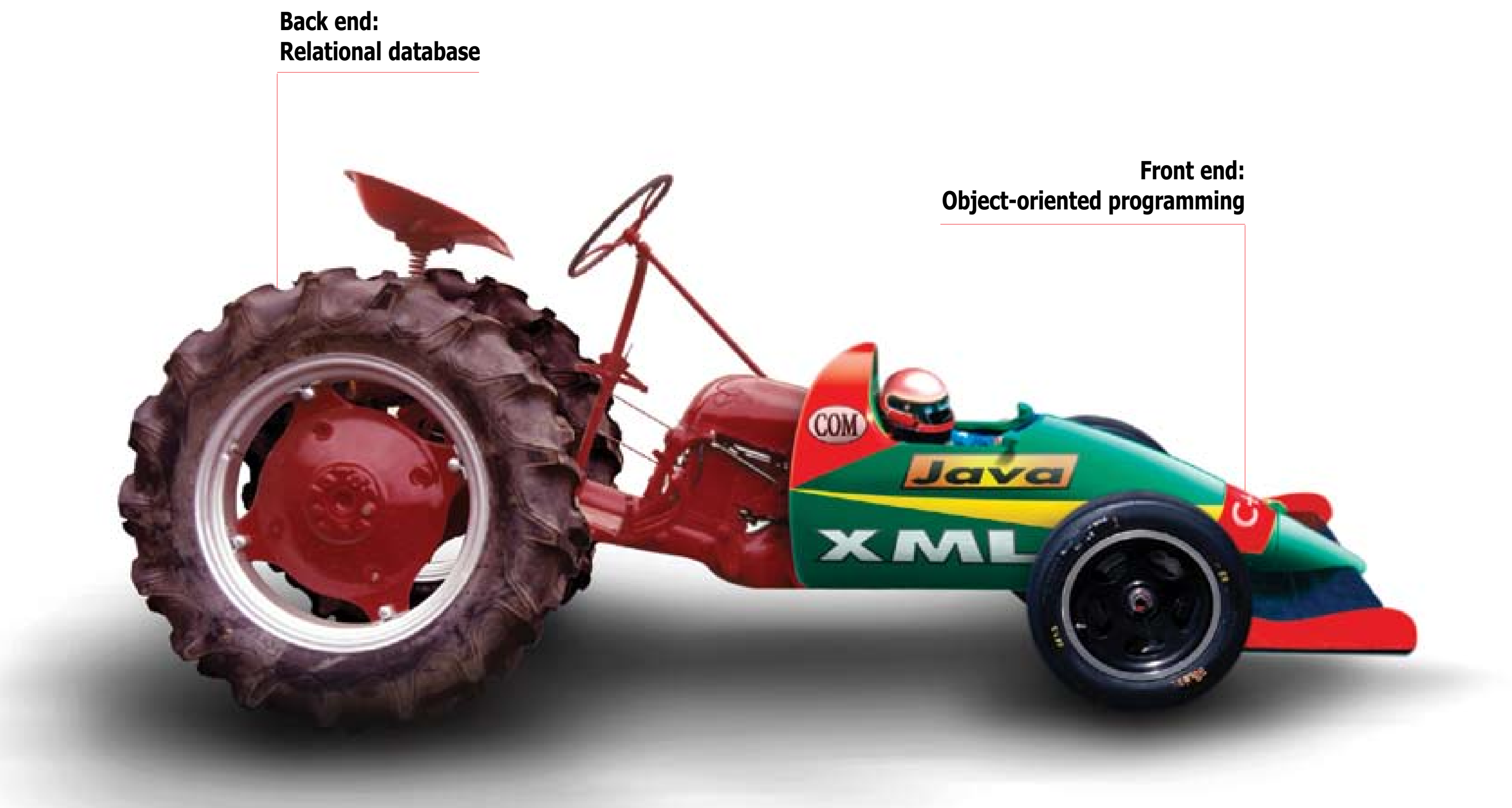


Figure 2 Enabled and disabled buttons

Innovations by InterSystems

You program in Java, but still use a relational database.



There's something wrong with this picture.

Unlike relational databases, Caché is a perfect match as the back end for object-oriented programming. It's the world's fastest object database, and runs SQL queries up to 5 times faster than relational databases. Plus, with an innovation by InterSystems called Jalapeño™, Caché persists Java objects without relational mapping – reducing development time for Java programmers by as much as 40%.

Caché delivers massive scalability on minimal hardware, requires little administration, and incorporates a rapid Web application development environment. It's available for Unix, Linux, Windows, Mac OS X, and OpenVMS – and is deployed on more than 100,000 systems ranging from two to over 50,000 users.

We are InterSystems, a global software company with a 28-year track record of innovations that enrich applications.

InterSystems CACHÉ®

Try an innovative database for free: Download a fully functional, non-expiring copy of Caché, or request it on CD, at **www.InterSystems.com/Cache21P**

```
public String getChosenItem()
public void setChosenItem( String items )
public void move( List sourceList,
   String[] sourceValues, List destList )
    ...
```

The "languageList" and "chosenItem" fields and the "move" method are accessed from the "execute" method of the form's submit action. In the Struts implementation, this is an instance of the Struts Action class.

```
/**
 * Populate the lists from the hidden field.
 * <p>
 * @param  mapping   action mapping
 * @param  form      action form
 * @param  request   HTTP servlet request
 * @param  response  HTTP servlet reponse
 * @throws Exception
 */
public ActionForward execute( ActionMapping mapping,
   ActionForm form, HttpServletRequest request,
   HttpServletResponse response ) throws Exception
{
   // Populate available list from language list.
   ExampleForm eForm = ( ExampleForm )form;
   eForm.getAvailableList().clear();
   eForm.getAvailableList().addAll(
      eForm.getLanguageList());

   // Populate chosen list from hidden field.
   eForm.getChosenList().clear();
   eForm.move( eForm.getAvailableList(),
      eForm.getChosenItem().split( "\\|" ),
      eForm.getChosenList());
}
```

JavaServer Faces Implementation

What is needed to implement this design pattern in JavaServer Faces? JSF is designed by some of the same people who designed Struts so we hope for a smooth migration path.

In JavaServer Faces, the JSP for the two lists is smaller, because JSF's tags are more compact. Instead of <table>, <tr>, and <td>, JSF uses <h:panelGrid>. Instead of <fmt:message>, JSF uses <h:outputText>. Instead of <html:select> and <html:optionsCollection>, JSF uses <h:selectManyListbox> and <f:selectItems>.

```
<h:panelGrid columns="3" rowClasses="center">
   <h:outputText value="#{bundle.available}" />
   <h:outputText value="" />
   <h:outputText value="#{bundle.chosen}" />
   <h:selectManyListbox style="width:100px; height:120px;"
      id="available" value="#{example.availableValues}"
      onchange="doUpdate( false, true );">
      <f:selectItems value="#{example.availableList}"/>
   </h:selectManyListbox>
      ...
   <h:selectManyListbox style="width:100px; height:120px;"
      id="chosen" value="#{example.chosenValues}"
      onchange="doUpdate( true, false );">
      <f:selectItems value="#{example.chosenList}"/>
   </h:selectManyListbox>
</h:panelGrid>
```

As in the Struts implementation, the <h:selectManyListbox> tags refer to the "availableList," "availableValues," "chosenList," and "chosenValues" properties of the backing JavaBean. The interfaces for these properties are identical to the Struts implementation. Internally the Struts Bean stores List properties as LabelValueBeans, while the JSF Bean stores them as SelectItems.

Like the Struts implementation, the four buttons in the middle column are implemented with standard HTML <input> tags. For JSF these have to be enclosed in <f:verbatim> tags. Other than that, the buttons are almost identical with the Struts implementation. The only wrinkle is that JSF generates hierarchical element identifiers. That's why JavaScript for JSF often contains identifiers like those with the "form:" prefix in the code below.

```
<h:panelGrid columns="1">
   <f:verbatim>
      <input type="button" style="width:100px;"
         id="add" onclick=
         "doMove( 'form:available', 'form:chosen', false );"
         value="<fmt:message key='add'/>"    /><br />
      <input type="button" style="width:100px;"
         id="addAll" onclick=
         "doMove( 'form:available', 'form:chosen',  true );"
         value="<fmt:message key='addAll'/>" /><br />
      <br />
      <input type="button" style="width:100px;"
         id="remove" onclick=
         "doMove( 'form:chosen', 'form:available', false );"
         value="<fmt:message key='remove'/>" /><br />
      <input type="button" style="width:100px;"
         id="removeAll" onclick=
         "doMove( 'form:chosen', 'form:available',  true );"
         value="<fmt:message key='removeAll'/>" />
   </f:verbatim>
   <h:inputHidden id="chosenItem"
      value="#{example.chosenItem}" />
</h:panelGrid>
```

Other than the hierarchical identifiers, the "doMove()" and "doUpdate()" functions for JSF are identical to those in the Struts implementation. JavaScript provides client-side interactivity in JSF just as it does in Struts.

A convenience of JavaServer Faces is its handling of the submit action. In JavaServer Faces, you can define the submit button to explicitly invoke a method in the form bean:

```
<h:commandButton action="#{example.submit}"
   value="#{bundle.submit}" />
```

The "example:submit" method reads the hidden field and populates the lists. Because this method is attached to

MAKE ANSWERS TO PERFORMANCE PROBLEMS COME TO YOU.

OPNET Panorama offers powerful analytics for rapid troubleshooting of complex J2EE/.NET applications. Panorama quickly identifies how application, web, and database servers are impacting end-to-end performance. With Panorama, you can pinpoint the source of a problem, so time and money aren't spent in the wrong places.

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

OPNET Panorama™ Real-Time Application Analytics

www.opnet.com/pinpoint

OPNET® Making Networks and Applications Perform™

OPNET Technologies, Inc. 7255 Woodmont Avenue, Bethesda, Maryland 20814 phone: (240) 497-3000 • e-mail: info@opnet.com • NASDAQ: OPNT the submit button, there's no need to implement an Action object.

```
public String submit()
{
   // Populate the available list from the language list.
   availableList.clear();
   availableList.addAll( languageList );

   // Populate the chosen list from the hidden field.
   chosenList.clear();
   move( availableList,
      chosenItem.split( "\\|" ), chosenList );
      ...
   return( "success" );
}
```

There's not much of a learning curve to JSF. Tags are different, but they usually produce smaller JSPs. Java code is similar and sometimes requires fewer objects.

The strongest advantage of JavaServer Faces is its component architecture. If you get the free download of Sun's Java Studio Creator, you'll find it contains a complete Add-and-Remove component that you can drag-and-drop in your GUI. Sun's component includes features such as "Move Up" and "Move Down" buttons to tweak the order of the chosen items. We can expect many such useful components to emerge as JSF development advances.

Conclusion

This article has described a standard UI design pattern for making ordered selections and selections from long lists. Implementations of this pattern were compared using Jakarta Struts and JavaServer Faces.

JSF provides a natural migration path for projects moving from Struts. The JSP tags are simplified; the backing bean code is similar; and if you need JavaScript for interactivity, the same functions can be used.

JSF can be thought of as a simplified, componentized version of Struts. Its designers have done exactly the type of work one hopes for in a "second system": they've added useful features and reduced complexity.

For any new Web project, JavaServer Faces should be strongly considered. For existing Struts projects, JSF provides a smooth migration path.

Resources

- Apache Software Foundation. http://struts.apache.org/, 2006.
- Steve Aube. A Dual Listbox Selection Manager. http://www.codeguru.com/Cpp/controls/listbox/article.php/c4755.
- Hans Bergsten. JavaServer Faces. O'Reilly. Sebastopol, CA. 2004.
- Heman Robinson. "Struts and JavaServer Faces: Design Patterns for List Selection." Java Developer's Journal, 11:3, 2006.
- Sun MicroSystems, Inc. Java Look and Feel Design Guidelines: Advanced Topics. Addison-Wesley Professional. New York. 2002.
- Sun MicroSystems, Inc. "JavaServer Pages Standard Tag Library." http://java.sun.com/products/jsp/jstl/.
- Sun MicroSystems, Inc. http://developers.sun.com/prodtech/javatools/jscreator/reference/faqs/technical/webforms/js_client_identifier.html.
- Sun Microsystems, Inc. Java Studio Creator. http://developers.sun.com/prodtech/javatools/jscreator/.
- Jennifer Tidwell. Designing Interfaces. O'Reilly. Sebastopol, CA. 2005.
- World Wide Web Consortium. HTML 4.01 Specification. http://www.w3.org/TR/html401/interact/forms.html#h-17.13. 1999.
- Weinschenk, Jamar, and Yeo, GUI Design Essentials. Wiley & Sons. New York. 1997.

"For any new Web project, JavaServer Faces should be strongly considered. **For existing Struts projects, JSF provides a smooth migration path."**

SAP TECHED '06: MEET. LEARN. INVENT.

Bringing the SAP worldwide community together.

Come to SAP TechEd '06 and learn how to transform existing business processes and IT landscapes and take advantage of the power and flexibility of enterprise service-oriented architecture. SAP TechEd '06 offers you the ability to meet and talk with SAP tech leaders and experts as well as offering you educational sessions, instructor-led hands-on workshops, and networking opportunities that keep IT managers, administrators, business analysts, project managers, and software developers focused on innovation and staying ahead of business change.

September 12 – 15 ▪ Las Vegas
October 5 – 6 ▪ Tokyo
October 18 – 20 ▪ Amsterdam
November 8 – 10 ▪ Bangalore

For more information visit us at
www.sapteched.com

06 SAP TECHED

Listing 1 – Jakarta Struts JSP

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
   "http://www.w3.org/TR/html4/strict.dtd">
<%@ taglib uri="/WEB-INF/tlds/struts-bean.tld"
   prefix="bean" %>
<%@ taglib uri="/WEB-INF/tlds/struts-html.tld"
   prefix="html" %>
<%@ taglib uri="/WEB-INF/tlds/struts-logic.tld"
   prefix="logic" %>
<%@ taglib uri="/WEB-INF/tlds/c.tld"
   prefix="c" %>
<%@ taglib uri="/WEB-INF/tlds/fmt.tld"
   prefix="fmt" %>

<HTML>
<HEAD>
   <TITLE>Add-and-Remove Pattern</TITLE>
   <link rel="stylesheet" type="text/css" href=
      '<%= request.getContextPath() + "/stylesheet.css" %>'>
</HEAD>

<BODY BGCOLOR="white" onload="doUpdate( false, false );">
<fmt:setBundle basename="com.kowaldesign.example.example"/>

<html:form action="/exampleWrite.do">

<table border="0" cellpadding="0" cellspacing="5">

   <%-- Add-and-Remove Pattern --%>
   <tr align="middle" valign="center">
      <td>
         <fmt:message key="available"/><br />
         <html:select property="availableValues"
            multiple="true" size="7" style="width:80px;"
            styleId="available"
            onchange="doUpdate( false, true );">
         <html:optionsCollection property="availableList"/>
         </html:select>
      </td>
      <td>
         <br />
         <input type="button" style="width:100px;" id="add"
            onclick="doMove( 'available','chosen', false );"
            value="<fmt:message key='add'/>"     /><br />
         <input type="button" style="width:100px;" id="addAll"
            onclick="doMove( 'available','chosen',  true );"
            value="<fmt:message key='addAll'/>" /><br />
         <br />
         <input type="button" style="width:100px;" id="remove"
            onclick="doMove( 'chosen','available', false );"
            value="<fmt:message key='remove'/>" /><br />
         <input type="button" style="width:100px;"
            id="removeAll"
            onclick="doMove( 'chosen','available',  true );"
            value="<fmt:message key='removeAll'/>" />
         <html:hidden styleId="chosenItem"
            property="chosenItem" />
      </td>
      <td>
         <fmt:message key="chosen"/><br />
         <html:select property="chosenValues"
            multiple="true" size="7" style="width:80px;"
            styleId="chosen"
            onchange="doUpdate( true, false );">
            <html:optionsCollection property="chosenList"/>
         </html:select>
      </td>
   </tr>

   <%-- Submit button --%>
   <tr align="middle" valign="top">
      <td colspan="3">
         <input type="submit"
            value="<fmt:message key='submit'/>" />
      </td>
   </tr>
</table>

</html:form>

<script language="JavaScript">
<!--

   /**
    * Move selected items between lists.
    * <p>
    * @param  sourceId  ID of source list
    * @param  destId    ID of destination list
    * @param  all       true iff moving all
    */
   function doMove( sourceId, destId, all )
   {
      // Move the items between the lists.
      var sourceElem = document.getElementById( sourceId );
      var destElem = document.getElementById( destId );
      for( var i = 0; ( i < sourceElem.length ); )
      {  if( sourceElem.options[ i ].selected || all )
         {  var newOption =
               document.createElement( "OPTION" );
            newOption.text = sourceElem.options[ i ].text;
            newOption.value = sourceElem.options[ i ].value;
            destElem.options[ destElem.length ] = newOption;
            sourceElem.remove( i );
         }
         else
            i++;
      }

      // Update the button states.
      doUpdate( false, false );

      // Store the chosen items in a hidden field.
      var chosenItem =
         document.getElementById( "chosenItem" );
      var chosenList = document.getElementById( "chosen" );
      chosenItem.value = "";
      for( var i = 0; ( i < chosenList.length ); i++ )
      {  chosenItem.value +=
            chosenList.options[ i ].value + '|';
      }
   }

   /**
    * Update the button states based on whether
    * lists have contents and selected items.
    * Deselect list items if requested to ensure
    * at most one list contains selections.
    * <p>
    * @param  offAvailable  deselecting available list
    * @param  offChosen     deselecting chosen list
    */
   function doUpdate( offAvailable, offChosen )
   {
      // Get the lists and deselect if requested.
      var availableList =
         document.getElementById( "available" );
      var chosenList =
         document.getElementById( "chosen" );
      if( offAvailable )
         availableList.selectedIndex = -1;
      if( offChosen )
         chosenList.selectedIndex = -1;

      // Update the button states.
      document.getElementById( "addAll" ).disabled =
         ( availableList.length == 0 );
      document.getElementById( "removeAll" ).disabled =
         ( chosenList.length == 0 );
      document.getElementById( "add" ).disabled =
         ( availableList.selectedIndex < 0 );
      document.getElementById( "remove" ).disabled =
         ( chosenList.selectedIndex < 0 );
   }

// -->
</script>

</BODY>
</HTML>
<<End Listing 1.>>
```

Listing 2. JavaServer Faces JSP

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
   "http://www.w3.org/TR/html4/strict.dtd">
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="/WEB-INF/tlds/c.tld" prefix="c" %>
<%@ taglib uri="/WEB-INF/tlds/fmt.tld" prefix="fmt" %>

<HTML>
<HEAD>
   <TITLE>Add-and-Remove Pattern</TITLE>
   <link rel="stylesheet" type="text/css" href=
      '<%= request.getContextPath() + "/stylesheet.css" %>'>
</HEAD>

<BODY BGCOLOR="white" onload="doUpdate( false, false );">
<f:loadBundle basename="com.kowaldesign.example.example"
   var="bundle"/>
<fmt:setBundle basename="com.kowaldesign.example.example"/>

<f:view>
   <h:form id="form">
      <h:panelGrid columns="1" rowClasses="center">

      <%-- Add-and-Remove Pattern --%>
      <h:panelGrid columns="3" rowClasses="center">
         <h:outputText value="#{bundle.available}" />
         <h:outputText value="" />
         <h:outputText value="#{bundle.chosen}" />
       <h:selectManyListbox id="available"
               style="width:100px; height:120px;"
          value="#{example.availableValues}"
               onchange="doUpdate( false, true );">
          <f:selectItems
                  value="#{example.availableList}"/>
      </h:selectManyListbox>
      <h:panelGrid columns="1">
    <f:verbatim>
       <input type="button" style="width:100px;"
                    id="add" onclick="doMove(
                    'form:available','form:chosen',false );"
          value="<fmt:message key='add'/>"    />
                 <br />
       <input type="button" style="width:100px;"
                    id="addAll" onclick="doMove(
                    'form:available','form:chosen', true );"
          value="<fmt:message key='addAll'/>" />
       <br /><br />
       <input type="button" style="width:100px;"
                    id="remove" onclick="doMove(
                    'form:chosen','form:available',false );"
          value="<fmt:message key='remove'/>" />
                 <br />
       <input type="button" style="width:100px;"
                    id="removeAll" onclick="doMove(
                    'form:chosen','form:available', true );"
          value="<fmt:message key='removeAll'/>" />
    </f:verbatim>
      <h:inputHidden id="chosenItem"
              value="#{example.chosenItem}" />
      </h:panelGrid>
      <h:selectManyListbox id="chosen"
              style="width:100px; height:120px;"
         value="#{example.chosenValues}"
              onchange="doUpdate( true, false );">
         <f:selectItems value="#{example.chosenList}"/>
      </h:selectManyListbox>
      </h:panelGrid>

      <%-- Submit button --%>
      <h:panelGrid columns="1">
         <h:commandButton action="#{example.submit}"
            value="#{bundle.submit}" />
      </h:panelGrid>
      </h:panelGrid>
   </h:form>
</f:view>

<script language="JavaScript">
<!--
   /**
    * Move selected items between lists.
    * <p>
    * @param  sourceId  ID of source list
    * @param  destId    ID of destination list
    * @param  all       true iff moving all
    */
   function doMove( sourceId, destId, all )
   {
      // Move the items between the lists.
      var sourceElem = document.getElementById( sourceId );
      var destElem = document.getElementById( destId );
      for( var i = 0; ( i < sourceElem.length ); )
      {  if( sourceElem.options[ i ].selected || all )
         {  var newOption =
               document.createElement( "OPTION" );
            newOption.text = sourceElem.options[ i ].text;
            newOption.value = sourceElem.options[ i ].value;
            destElem.options[ destElem.length ] = newOption;
            sourceElem.remove( i );
         }
         else
            i++;
      }

      // Update the button states.
      doUpdate();

      // Store the chosen items in a hidden field.
      var chosenItem =
         document.getElementById( "form:chosenItem" );
      var chosenList =
         document.getElementById( "form:chosen" );
      chosenItem.value = "";
      for( var i = 0; ( i < chosenList.length ); i++ )
      {  chosenItem.value +=
            chosenList.options[ i ].value + '|';
      }
   }

   /**
    * Update the button states based on whether
    * lists have contents and selected items.
    * Deselect list items if requested.
    * <p>
    * @param  offAvailable  deselecting available list
    * @param  offChosen     deselecting chosen list
    */
   function doUpdate( offAvailable, offChosen )
   {
      // Get the lists and deselect if requested.
      var availableList =
         document.getElementById( "form:available" );
      var chosenList =
         document.getElementById( "form:chosen" );
      if( offAvailable )
         availableList.selectedIndex = -1;
      if( offChosen )
         chosenList.selectedIndex = -1;

      // Update the button states.
      document.getElementById( "addAll" ).disabled =
         ( availableList.length == 0 );
      document.getElementById( "removeAll" ).disabled =
         ( chosenList.length == 0 );
      document.getElementById( "add" ).disabled =
         ( availableList.selectedIndex < 0 );
      document.getElementById( "remove" ).disabled =
         ( chosenList.selectedIndex < 0 );
   }

// -->
</script>
</BODY>
</HTML>
<<End Listing 2.>>
```

Detecting J2EE Problems Before They Happen

Derived Model Analysis

by Alan West & Gordon Cruickshank

This article introduces a new form of analysis for Java EE applications: a runtime abstract application model derived automatically from an application server using stored knowledge of Java EE construction. The model is used dynamically to do extensive automatic checks for a range of construction errors that could produce poor performance or unreliability. The model also lets server behavior be dynamically visualized in real-time or retrospectively.

There has been a lot of attention given lately to the topic of of Model Driven Architecture (MDA), which aims to create working systems by generating source code from successively transformed high-level component models. While doubts have been cast on the real-world robustness of this idea — and previous code-generation solutions haven't been a big success — there's no doubt that the possibility of working with software at a more abstract level holds a strong appeal for engineers.

Although the inauspicious history of CASE tools suggests that making a project dependent on model-driven code generation could be limiting, the central tenet of MDA — the ability to view and analyze our application at an abstract level — is a powerful and attractive goal. Even if our application grew beyond an initial set of predefined patterns and code templates we'd still like to be able to validate and understand it based on a design-level description of its operation.

Derived Model Analysis (DMA)

If we don't have a predefined model, how are we going to get one? Well, if you try to describe your application to someone else you'll almost certainly use architecture-level abstractions: the services it uses; the main business and data components and how these relate. So it would be good if similar high-level abstractions could be derived and presented automatically by analyzing and monitoring the execution of your application. Model elements would include application components, the application server services they use, and the data access, transaction management, and calling relationships between them.

Once application model elements were identified they would be updated dynamically during execution. Monitoring the changing patterns of inter-relationships in the model would automatically detect construction-quality problems by detecting unlikely relationships, unnecessary and duplicated relationships, and undesirable model entity states. Instead of trying to spot problems in the clutter of source code we could see key abstractions directly in the model.

eoLogic terms this form of indirect application monitoring Derived Model Analysis (DMA): tools analyze Java EE applications both statically and during server execution to derive an abstract model that includes both application components and Java EE services. Subsequent changes to the model form a dynamic event sequence that can be used to (a) track and validate application execution and (b) visualize the model. Lower-level application execution details can be recorded in the context of the sequence of model changes.

Note that DMA is not a profiling technique – it doesn't aim to identify current code hotspots; instead, it analyses how services have been constructed and are being used. The idea is to identify places where hotspots or unreliability may occur under load. This deeper form of analysis can be used to find problems before they manifest themselves and without the application being loaded during testing. These problems include incorrect or inefficient transaction grouping, inefficient database access, unreliable sequences of inter-component communication, and failure to control service lifecycles correctly. There's no need to drive the application to a point at which it exhibits slowdown, and the results need little interpretation.

Deriving a DMA Model

To generate and validate an abstract model of an application a tool must be able to monitor events in the server and interpret them in light of the relevant stored knowledge.

This includes definitions of the main abstract entities we're interested in (transaction manager, transaction resources, transactions, EJB containers, JMS destinations, etc.), the possible relationships between these entities, and invalid and valid patterns of relationships and states. DMA forms them into an abstract Entity-Relationship-Attribute (ERA) model as the system executes, with model changes triggering annotated definitions of problem states.

Relationship to JMX

The model sounds a lot like Java Management Extensions (JMX) — which essentially define a form of abstract model for purposes of managing and monitoring Java applications, and it suggests that possibly DMA could be layered on top of the information available from JMX MBeans. In detail, what characteristics does a DMA model require?

- It must be an accurate and com-

Gordon Cruickshank is co-founder of eoLogic (http://www.eologic.com), a software tools company created to develop innovative testing and debugging solutions. He was previously development manager at Wind River Systems and Objective Software Technology, building C++ debugging and object visualization tools.

Alan West is CTO of eoLogic (http://www.eologic.com), responsible for all product development. He was previously a founder of Object Software Technology Ltd, and has over 20 years experience in software tool design and architecting large software systems.

plete abstract model of an application, linking static (source) and runtime application components.
- It must be able to be updated in real-time as the server executes generating meaningful sequential event flows.
- It must support a wide range of relationship types including application-level call relationships.
- It must be able to be intimately combined with knowledge about valid and potentially invalid model forms.
- It must be possible to relate model-level information easily back to application source.
- It must be easily filtered to focus on different aspects of server operation.
- And it must be easily and intuitively understood.

JMX goes some way towards what is needed: It provides an abstract model of an application for both its static and dynamic aspects; it allows easy selection of MBeans; many MBeans relate directly to easily understood aspects of server operation; there's a notification system for attribute changes and there's even an MBean relation service.

However, for our purposes it also has some serious limitations. Many of the relationships we have to monitor are based on calling sequences and application component relationships. Designed primarily for system management and threshold monitoring, JMX doesn't provide the source-level monitoring and mapping that the detection and (especially) the explanation of application construction errors requires. Also, the level of coverage is generally insufficiently detailed to provide a coherent execution model for the purposes of visualization. And if we want to use the product to investigate problems requiring the ability to freeze the server at the point of problem detection and extract stack and related data information then JMX isn't precise enough.

So the approach that we adopted is to create a more detailed runtime ERA model specialized for the following purposes:
- Representing sequences of server operation precisely and clearly
- Detecting construction errors based on component interrelations, including call sequences and transaction membership
- Explaining construction errors by relating model entities and relationships to precise source references
- Providing an intuitive visual model of sequential server operation
- Supporting model tracing and playback
- Supporting integrated debugging

This specialized model then provides the structure for attaching knowledge about model entity roles and valid and invalid patterns of model relationships and attributes, together with details on problem descriptions and suggested fixes.

The need for detailed tracking of calls and object states means that the

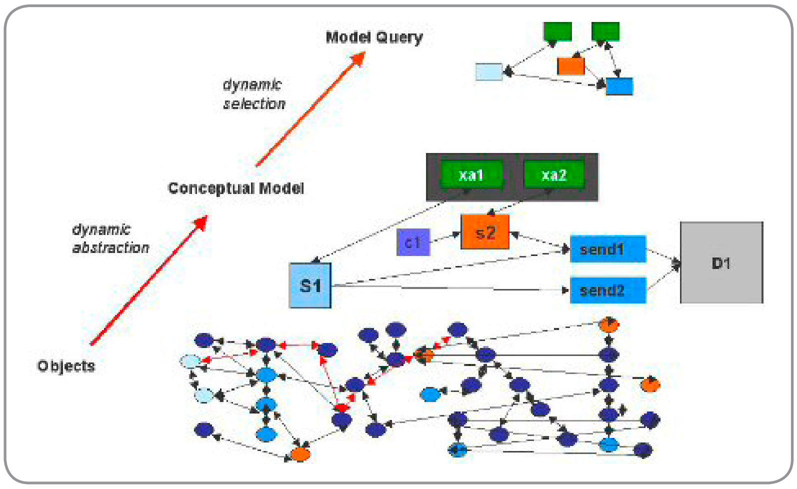
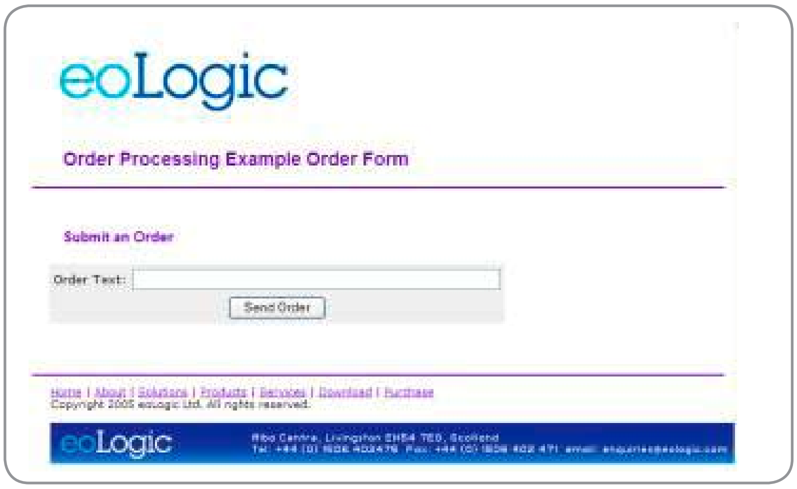


Figure 1 DMA operation by abstraction and selection


eoLogic

Order Processing Example Order Form

Submit an Order

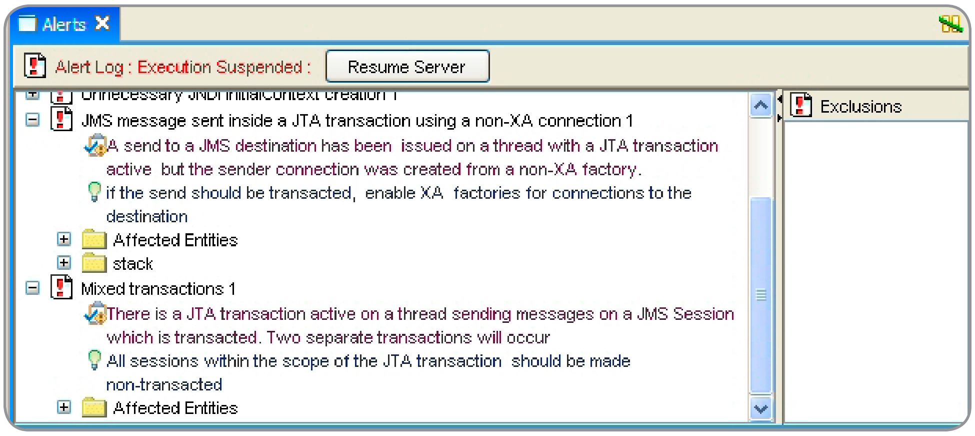
Order Text:

Send Order

Home | About | Solutions | Products | Services | Download | Purchase
Copyright 2005 eoLogic Ltd. All rights reserved.

eoLogic Alba Centre, Livingston EH54 7EG, Scotland
Tel: +44 (0) 1506 402476 Fax: +44 (0) 1506 402 471 email: enquiries@eologic.com

Figure 2 Example order form


Alerts

Alert Log : Execution Suspended : Resume Server

Unnecessary JNDI InitialContext creation 1

JMS message sent inside a JTA transaction using a non-XA connection 1

A send to a JMS destination has been issued on a thread with a JTA transaction active but the sender connection was created from a non-XA factory.

if the send should be transacted, enable XA factories for connections to the destination

Affected Entities

stack

Mixed transactions 1

There is a JTA transaction active on a thread sending messages on a JMS Session which is transacted. Two separate transactions will occur

All sessions within the scope of the JTA transaction should be made non-transacted

Affected Entities

Exclusions

Figure 3 Use case alerts

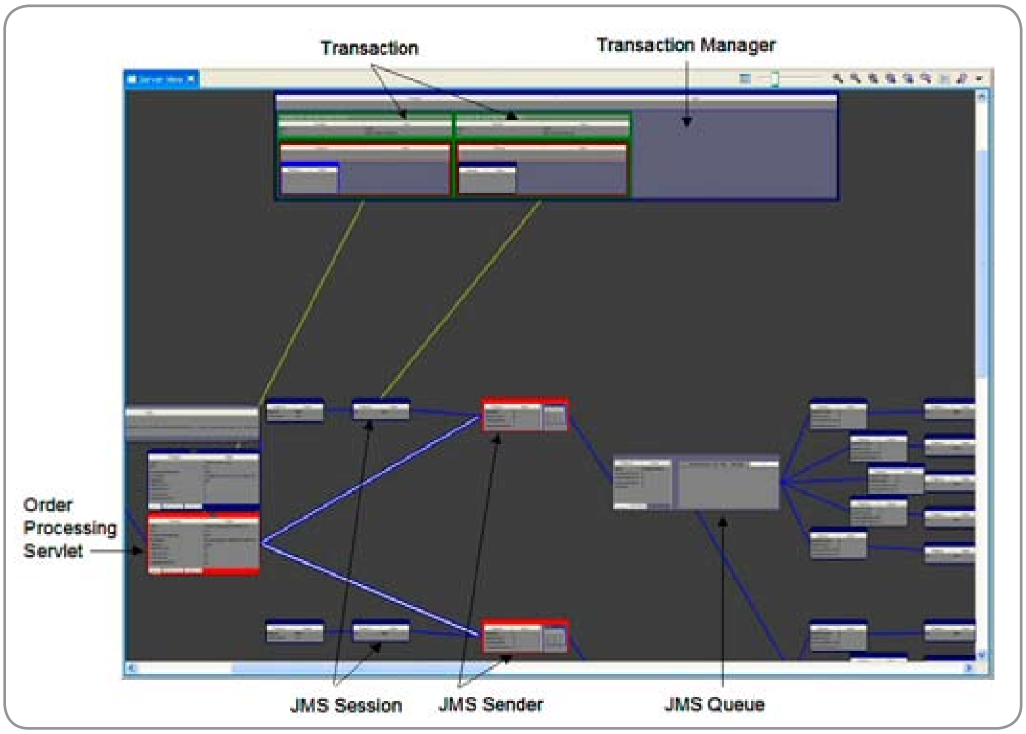


Figure 4 Entities and relationships in the server view for the mixed transactions alert

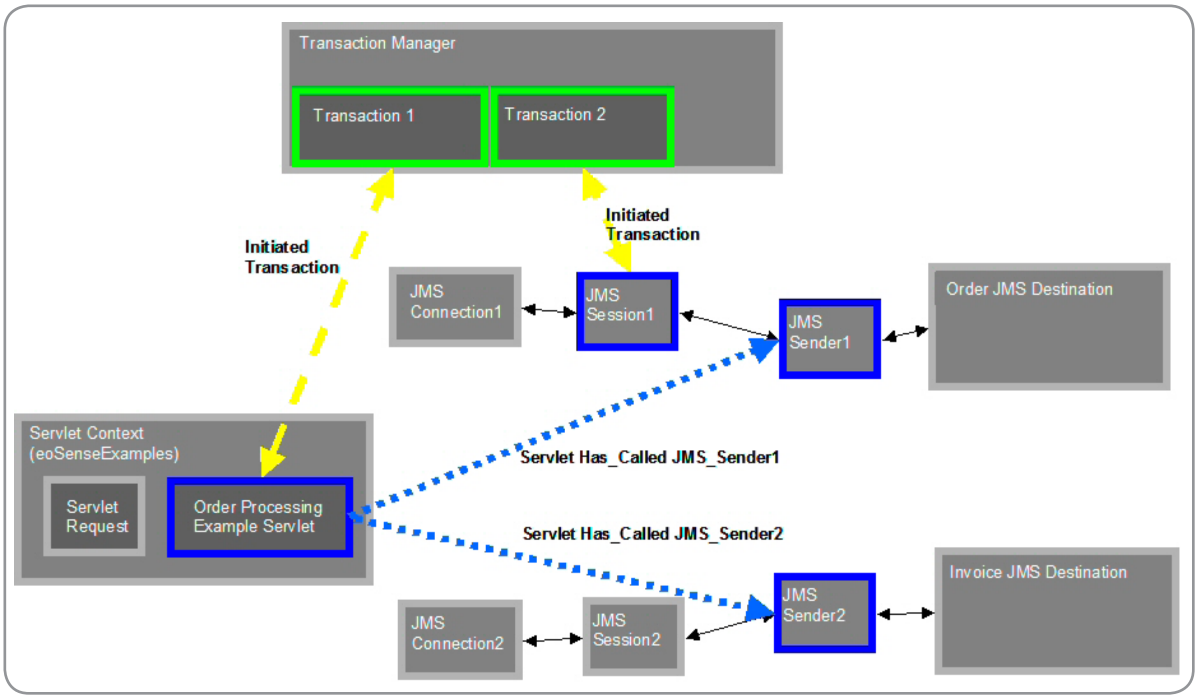


Figure 5 Diagram of entities and relationships at the mixed transactions alert

DMA engine moves from the realm of JMX and more towards an application of Aspect-Oriented Programming (AOP), combining the planned abstraction of JMX with the detailed and flexible monitoring and intervention of AOP. Having said this, it would be wasteful not to exploit the JMX information provided by a server. Some JMX MBeans serve as important internal DMA monitoring and access points, but are augmented with additional monitoring and updating points in the server.

DMA Error Detection

As shown in Figure 1 DMA abstracts from the underlying framework and application objects to a conceptual ERA model. Queries against this model then provide the means for problem recognition.

Fiorano SOA™ 2006

The Quickest path to an SOA

- **FioranoMQ™ 2006 –** world's fastest, most scalable JMS
- **Fiorano ESB™ 2006 –** CAD/CAM for distributed applications
- **Fiorano BPEL Server –** simplifying business process orchestration
- **Fiorano Tools –** BPEL Studio, Mapper, FEPO, etc
- **Fiorano Components –** 60+ pre-built adapters

Benefits

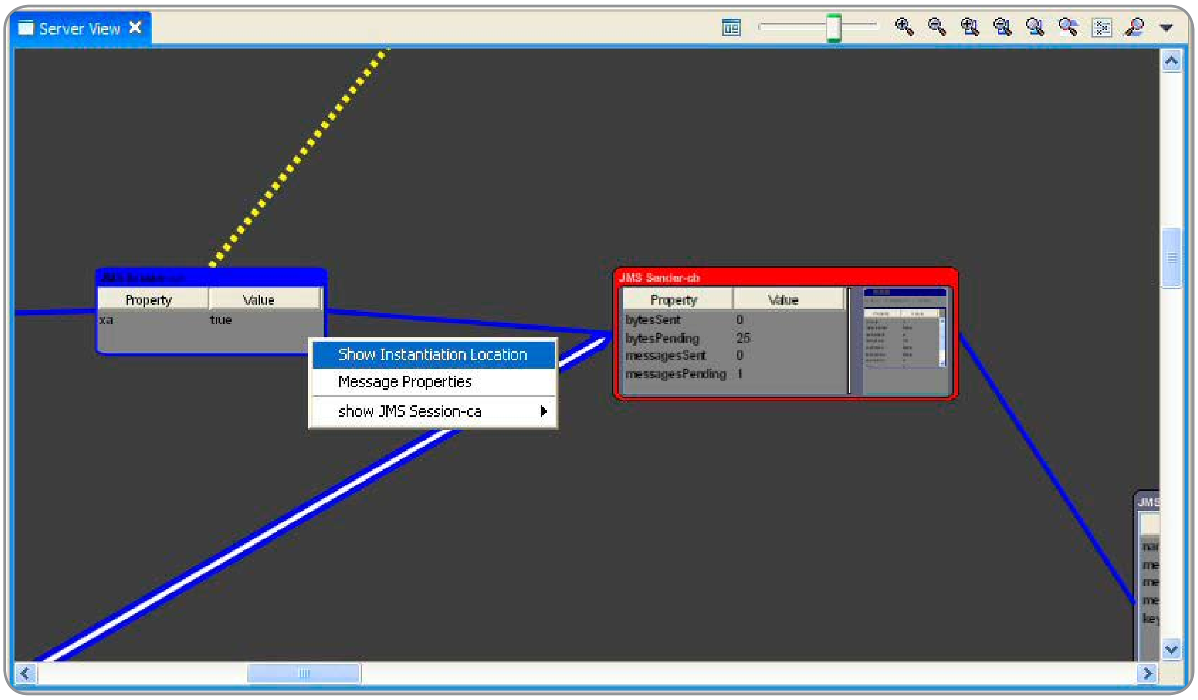
- **Adherence to popular industry standards** - JMS, COM, .NET, JCA, JMX, BPEL, SOAP, etc.
- **Multi-language, Multi-platform, Multi-protocol**
- **Unmatched Scalability and High Performance**
- **Quick, Measurable ROI**

Download your copy of Fiorano today!

www.fiorano.com/downloadsoa

Fiorano®

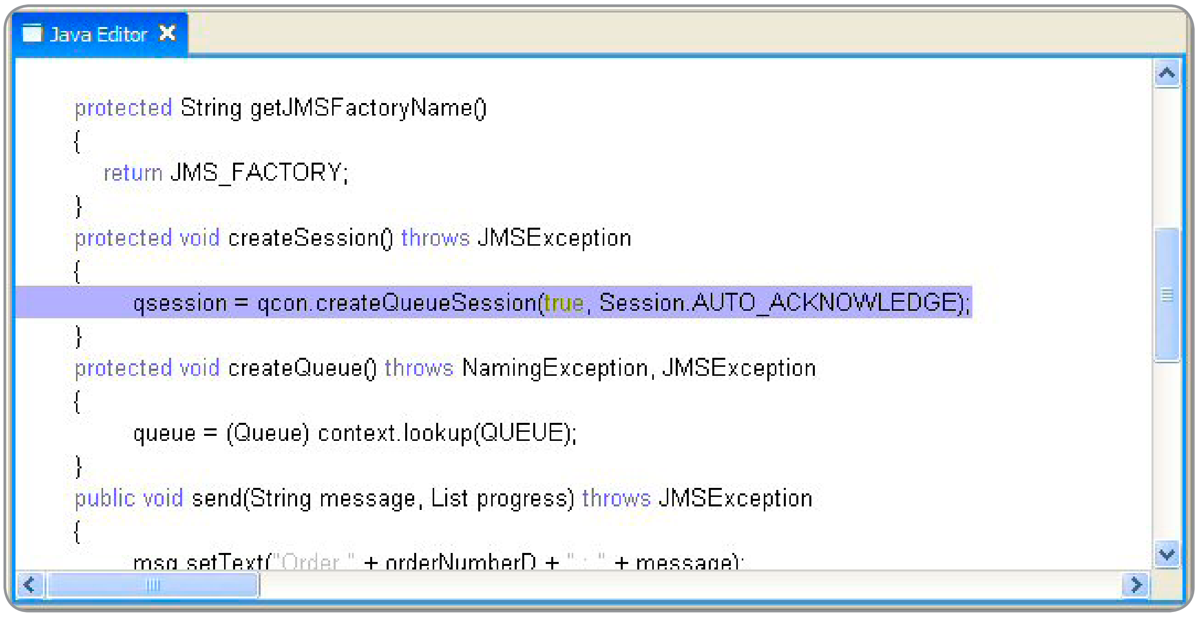
Enabling Change at the speed of thought



Property	Value
xa	true

Property	Value
bytesSent	0
bytesPending	25
messagesSent	0
messagesPending	1

Figure 6 The instantiation location for the JMS Session



```
protected String getJMSFactoryName()
{
    return JMS_FACTORY;
}
protected void createSession() throws JMSException
{
        qsession = qcon.createQueueSession(true, Session.AUTO_ACKNOWLEDGE);
}
protected void createQueue() throws NamingException, JMSException
{
        queue = (Queue) context.lookup(QUEUE);
}
public void send(String message, List progress) throws JMSException
{
        msg.setText("Order " + orderNumber() + " : " + message);
```

Figure 7 JMS Session creation source line

Usually the abstraction stage is primarily one of selection as key objects are monitored, but it can also require composition of elements from more than one underlying object.

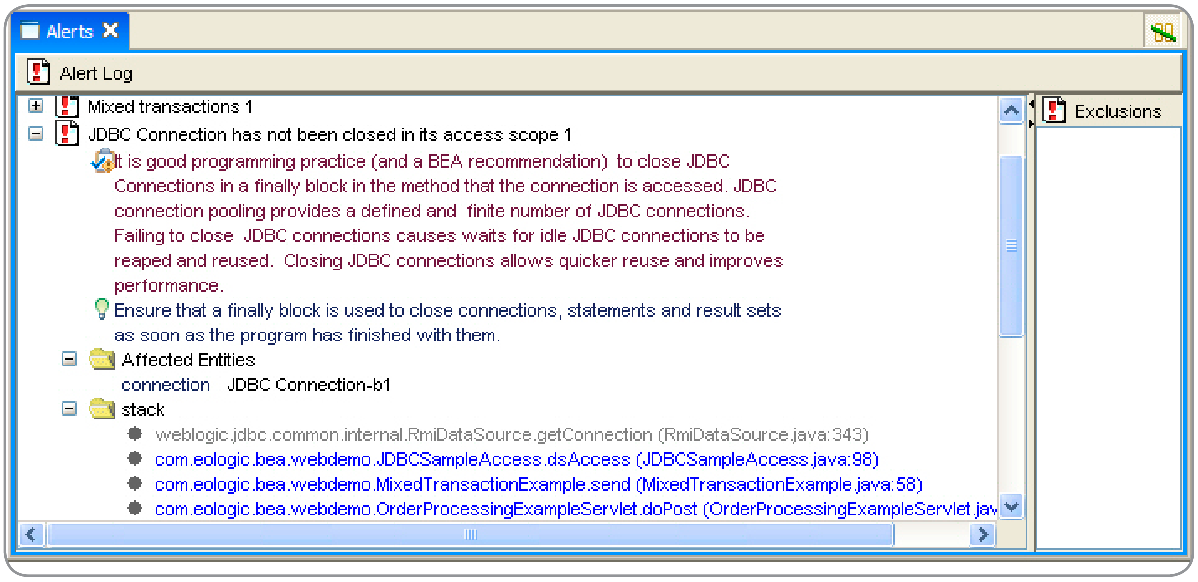
DMA Use Case

To look at how the abstraction mechanisms of DMA allow construction problems to be detected and explained let's look at it operating on a sample application. We'll aim to show how we can identify a pattern of application and framework components that indicates a problem. We'll then show how the problem can be visualized and explained back to the source level by exploring the model at the point of detection.

Example Application

Figure 2 shows a simple Web-based order-processing example that accepts orders and processes them in the following way:

- An order invoice is created and queued to an existing invoice service using JMS to create and process the invoice.
- The order details are queued to an



```
⊞ Mixed transactions 1
⊟ JDBC Connection has not been closed in its access scope 1
    It is good programming practice (and a BEA recommendation) to close JDBC
    Connections in a finally block in the method that the connection is accessed. JDBC
    connection pooling provides a defined and finite number of JDBC connections.
    Failing to close JDBC connections causes waits for idle JDBC connections to be
    reaped and reused. Closing JDBC connections allows quicker reuse and improves
    performance.
    Ensure that a finally block is used to close connections, statements and result sets
    as soon as the program has finished with them.
  ⊟ Affected Entities
      connection   JDBC Connection-b1
  ⊟ stack
      ● weblogic.jdbc.common.internal.RmiDataSource.getConnection (RmiDataSource.java:343)
      ● com.eologic.bea.webdemo.JDBCSampleAccess.dsAccess (JDBCSampleAccess.java:98)
      ● com.eologic.bea.webdemo.MixedTransactionExample.send (MixedTransactionExample.java:58)
      ● com.eologic.bea.webdemo.OrderProcessingExampleServlet.doPost (OrderProcessingExampleServlet.jav
```

Figure 8

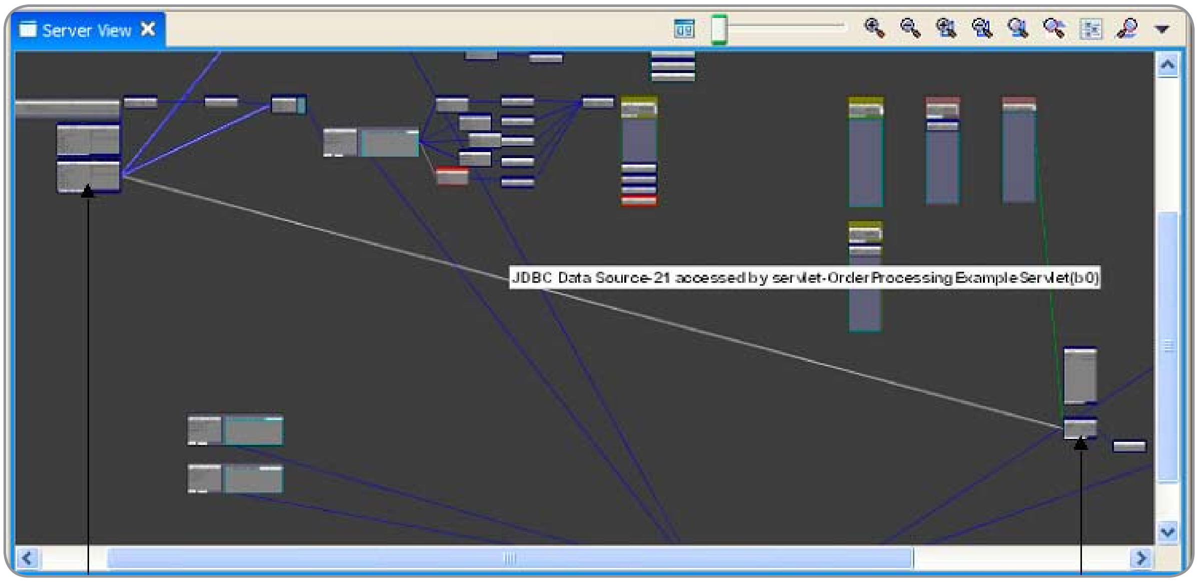

Server View
JDBC Data Source-21 accessed by servlet-OrderProcessingExampleServlet(b0)

Figure 9

order processing system using JMS to process and deliver the order separately.

However there's a problem: The invoices don't arrive at the invoice processing application although the order entries are processed correctly.

Monitoring the Application

To monitor the sample application we'll run the WebLogic server from our DMA analyzer called eoSense, which comprises a server agent and a client. The agent constructs and checks the abstract model as WebLogic executes. When a problem is detected, the agent signals an alert to the client.

Transaction-Related Alerts

Running the example application results in the initial alerts shown in Figure 3 being detected (after several less serious alerts).

Looking at the alerts in more detail, there was a:

- JMS Message sent inside a JTA Transaction using a non-XA Connection – A JMS Connection created from a non-XA Factory was used to create a JMS session and sender. The sender was then used with the context of a JTA transaction. This may indicate that an XA JMS Connection should have been used instead.
- Mixed Transactions – The JMS sender has been used from a JMS Session marked as transacted, but there's already a JTA transaction active on the current thread.

DMA Visualization

When the Mixed Transaction alert is recorded a diagram of the ERA model allows the context of the problem to be understood. In eoSense this is called the Server View and an image of the server view is shown in Figure 4.

We can see that there are two active transactions, one linked to the Order Processing Servlet and the other linked to a JMS Session. We can also see that the Order Processing Servlet has communicated with two JMS Senders. Figure 5 shows diagrammatically the named key entities and relationships from Figure 4.

- There are two in-flight transactions held by the Transaction Manager
- There is an "initiated By" relationship between the OrderProcessingExample Servlet and Transaction 1
- There is an "Initiated By" relationship between the JMS Session 1 and Transaction 2
- The OrderProcessingExample Servlet has sent two Messages: There is a "Has_Called" relationship to JMS Sender 1, which is attached to the Order JMS Destination, and a "Has_Called" Relationship to JMS Sender 2, which is attached to the Invoice JMS Destination.

Note: This example is not an endorsement of initiating JTA transactions in servlets. That's another doubtful practice – and one that eoSense can also detect – but it's simpler to show the example this way.

Problem Explanation

By examining the alerts, and model entities and relationships, we can identify the elements of the problem:

- There are two transactions, not one as expected
- One transaction was initiated by the servlet; the other by the JMS session. The second transaction is unexpected.

Now we can use the model to map the problem back down to the source level. By selecting the JMS Session, as shown in Figure 6, we can examine the point at which it was created (source creation points are available for application-level entities):

The displayed source in Figure 7 shows that it's been created with the transaction attribute set to true:

If we want both the order and invoice JMS messages to form part of the same overriding JTA transaction, the transaction attribute (the first argument) should be false. And because the JMS Session-specified transaction was unintended there's no code present in the application to commit it (as the presence of the transaction in the eoSense view after order entry processing confirms). And even if the gross error of not committing the JMS-initiated transaction hadn't happened we would still have code that incorrectly creates two transactions, with the possibility of intermittent damaging inconsistencies.

Mixed Transactions Model Alert Query

The DMA model consists of the dynamic set of abstract entities and their relationships, a small part of which we've already examined. But what model trigger fired to signal the Mixed Transactions alert? eoSense defines query triggers as plug-in script inserts to its database, but the trigger can be expressed concisely as:

```
when relation Transaction_InitiatedBy cre-
ated and relation.to.type == JMSSession and
exists( select Transaction t1 from
Transactions
where relation.from.attribute.thread ==
t1.attribute.thread)
```

Having identified the key transaction and session entities, we can then navigate to related entities to show the full set of entities and relationships in the scope of the problem and relate these back to source-level statements. In the example, these are the creators of the original transaction and the problematic session.

JDBC Connection Not Closed Alert

If we continue, Figure 8 shows the next alert that appears.

This alert indicates that a JDBC Connection has not been closed before the methods using it have returned indicating inefficient use. The problem connection is indicated, with captured stacks showing its use without closure, mapping the problem back to the source code.

Visualization

The image of the Server View in Figure 9 shows the direct access from the servlet to the DataSource (itself a doubtful practice which could be checked).

Connection not Closed Model Alert Query

This alert requires a model trigger to dynamically create a second trigger that then fires to indicate the problem:

```
when relation DataSource_Returned_
Connection created
create Trigger(return,single) t1 on
findFrame(method.returns(java.sql.
Connection) == false)
when t1
connection.attribute('closed') == false)
```

Note that this rule is aimed at detecting an unsafe pattern of use. If we just wanted to detect connection leaks we could track references but the intention in DMA checking is to highlight unsafe program construction patterns (as well as obviously faulty ones) and so achieve code that's not just less wasteful, but more maintainable.

Efficiency

Doesn't visualization and modeling of this kind impose far too heavy a performance penalty? Well, no. The model is derived from efficient, highly constrained monitoring in the application server, the extent of which is dynamically controlled. The frequency of client updates for visualization can be set to any desired level, and the model can be examined only in retrospect, if that is preferred.

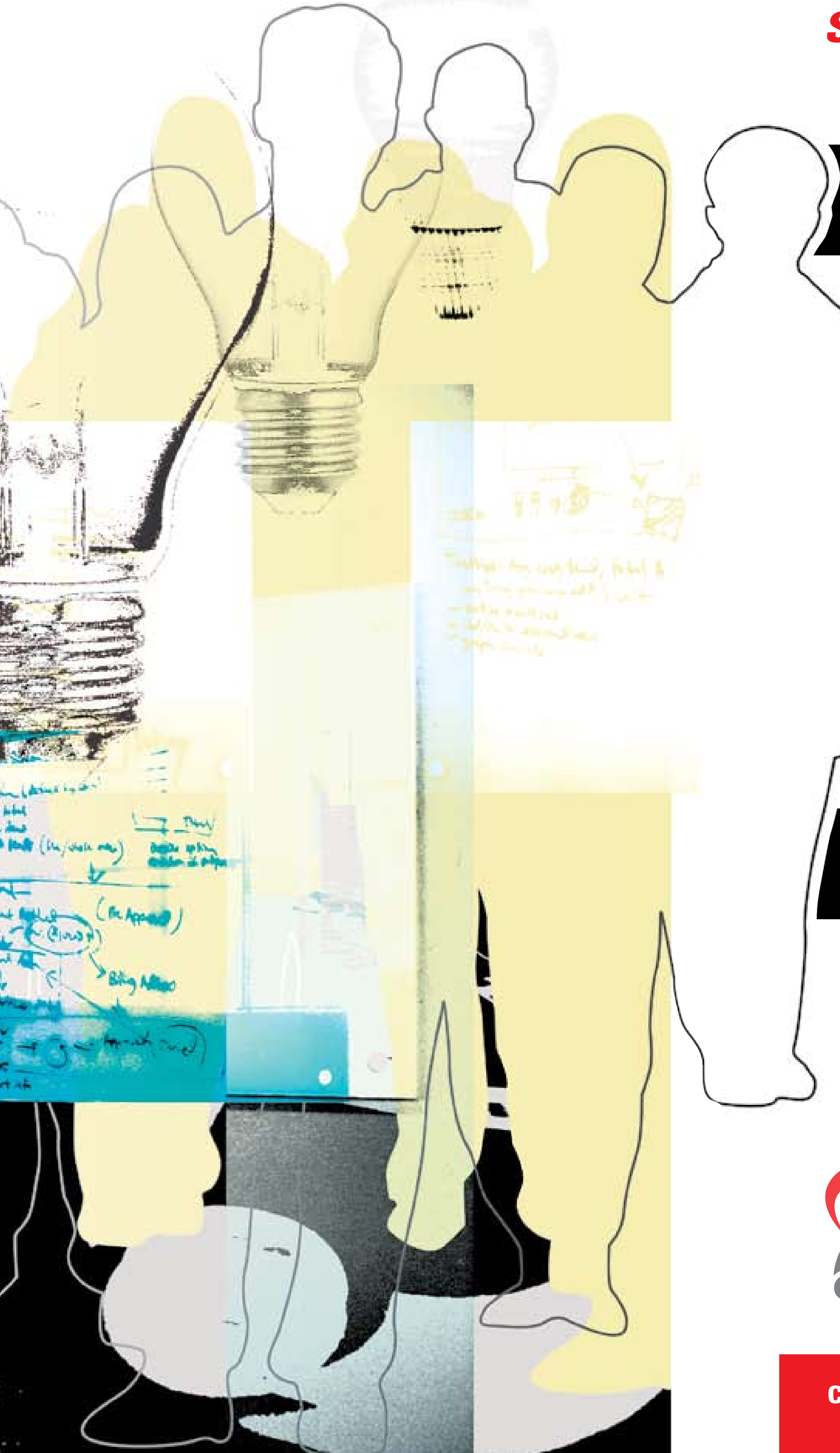
Summary

We've seen an example of Derived Model Analysis in action, deriving an entity-relationship model dynamically from an executing Java EE application, and using this to detect and, importantly, explain clearly serious structural problems that were not exhibiting any obvious effect and would not be obvious from the source code or from tracing application components.

eoSense can use DMA to represent visually almost all Java EE services and can monitor these independently or in combination. It can automatically detect a wide range of serious construction defects, with more detectors being added as new problem patterns are defined. It's clear that, as applications become increasingly based on standardized frameworks, automatically identifying design-level models from application execution and using those models to validate the applications becomes a real and powerful possibility.

Adobe | Solution PARTNER

I have the next great software idea.



I know how to change my industry.

I know people.

I know how to get investors on board.

I can lead.

I can inspire.

I need to do this.

YEAH, THAT'S ME.

Unlock your potential with the help of industry leaders in Rich Internet Application development. Discover how everyday we help people just like you at cynergysystems.com/thatsme.

CYNERGY

cynergysystems.com/thatsme

A Generic JMS Listener for Apache Axis 1.x

A needed transport-level handler

by Parameswaran Seshan

Unlike the HTTP protocol there's no stable default JMS listener for invoking the Web Services exposed in Apache Axis 1.x using JMS (Java Message Service) as the transport protocol – other than the one provided merely for demo purposes.

This article describes a fully working generic JMS listener that can act as a JMS transport receiver handler for Axis and allow service clients to uniquely address individual Web Services in a JMS way and invoke them over JMS.

Apache Axis is a popular Java-based Open Source platform for exposing Web Services. It has native support for handling invocations into Web Services based on the SOAP (Simple Object Access Protocol) application protocol. By default, the Axis server supports HTTP as the protocol for transporting the SOAP payload and provides an HTTP transport listener to do the same on its own. The HTTP transport listener accepts the SOAP requests coming over HTTP and then hands off the SOAP payload to the Axis engine for application-level handling of the request (like SOAP parsing, extracting input parameters, invoking the right service implementation, etc.), and gets the response SOAP message from the Axis engine and sends it back to the caller in the HTTP response.

For those enterprise applications where reliable invocation and guaranteed delivery of invocation messages are important, JMS, rather than HTTP, is the preferred protocol for transporting SOAP messages. JMS implementation providers, with built-in reliability features like re-try mechanisms, ensure that messages reach the message consumer application whatever the case. JMS is also the best way to handle asynchronous invocations of Web Services.

However, what Axis 1.x provides for JMS transport protocol use is only a basic demo listener that's not really meant for production-level use. This listener is also not easy to use and isn't flexible enough to be able to specify and handle unique endpoint addresses for each individual Web Service exposed. So, for client applications that need to invoke the Axis Web Services over JMS a flexible, stable, and easy-to-use JMS transport listener and handler is required.

This article implements a generic JMS listener and describes how it is to be used along with the Axis server. For purposes of this paper, I've considered the Open Source frameworks Apache Axis 1.2 and JMS provider OpenJMS 0.7.6.1. However, this should be largely applicable to the higher versions of Axis in the 1.x series too.

Parameswaran Seshan is a technical architect with Software Engineering and Technology Labs, the R&D division of Infosys Technologies Ltd. in Bangalore, India. His areas of expertise include Business Process Management systems, Web Services, and Java.

The Addressing Model

In an Axis server, each Web Service is described in the server-config.wsdd file and the service name mentioned there becomes part of the concrete HTTP URL (concrete port binding) for accessing the Web Service. For example, to access a "StockQuoteService" defined in the server-config.wsdd of the Axis server running in host www.samplehost.com, the default HTTP URL would be http://www.samplehost.com/axis/Services/StockQuoteService. Each Web Service similarly defined in server-config.wsdd will have a unique access URL like the one above with the first portion of the URL, i.e., http://www.samplehost.com/axis/Services/, remaining the same. In this sense, each Web Service will be addressed with a unique HTTP destination.

We can choose to follow a similar model for exposing the same Web Services over the JMS transport protocol. In other words, each service endpoint will be available at a unique JMS destination (aka a queue). So for each Web Service defined in the Axis server, we define a separate queue in the JMS provider – in our case OpenJMS – and update the openjms.xml file in the config folder of the OpenJMS home for defining one queue for each Axis-deployed Web Service that's meant to be accessible over JMS. In this article, the approach taken is to use the service name defined in Axis itself as the queue name (similar to the HTTP concrete binding mentioned above). For example, for the sample Web Service called "MessageService" provided in the Axis distribution, we can define a queue with the same name putting the entry <AdministeredQueue name="MessageService"/> in openjms.xml.

This addressing model applies uniformly to both RPC-style and message-style Axis services. It's more straightforward and standards-compatible with the WSDL specs. The JMS destination for each Web Service becomes the address in the concrete port binding for the service and the Web Service clients can directly use this concrete JMS destination mentioned in the WSDL file for invoking the service.

This is a better model than the non-standard way of specifying the Axis Web Service name as the prefix in the request SOAP message body's first XML element. For example, for the invocation of the method "getQuote" in the Axis sample Web Service named "urn:xmltoday-delayed-quotes," which is an RPC-style service, the basic JMS listener provided in the Axis distribution expects the client to create the SOAP body element <urn:xmltoday-delayed-quotes:getQuote> containing the service name as the XML prefix.

The client does this through code similar to call.setOperationName(new QName("urn:xmltoday-delayed-quotes:getQuote", "getQuote")). This fact doesn't appear in the WSDL definition of the service hence interoperability with different external service clients could become an issue. It's best to stick to the details given in the WSDL file and with the new service endpoint addressing scheme introduced here, all the clients can, in a standard way, just keep the first SOAP body element as <getQuote> following the WSDL details alone, thereby improving interoperability.

The Role of JMS Listener

The JMS listener is a server-side component that needs to listen to incoming JMS messages containing SOAP messages at the defined JMS queue. These SOAP messages are request messages coming from service clients that are trying to invoke the Web Service(s). Once the JMS message is received, the onMessage() method in the listener needs to get the message content which is the SOAP XML payload, and invoke the Axis engine supplying the SOAP XML message and also specifying to Axis the name of Web Service that's being invoked. This handing over of the SOAP message to Axis server is the key responsibility of this listener.

Then, if the client expects a response back from the Web Service (such as in a pseudo-synchronous call), the JMS listener needs to get the SOAP response message from the Web Service and put it as the payload in a new JMS message and send this message to the JMS queue destination the client is waiting on. This JMS listener can be used for receiving requests of both RPC-style and document-style Web Services invocations since it doesn't read and interpret the SOAP message at all; it just sticks to its role as a transport-level handler.

Implementing the Listener

Now we'll look at the implementation of the JMS listener. To realize the addressing model described above, the approach is to have one instance of the listener class GenericJMSSOAPListenerForAxis for each Axis Web Service. This class that implements javax.jms.MessageListener is a generic listener for JMS Web Service requests. I'll explain the salient parts of this class in this section and the next. The full source code of this and other classes discussed here is available for download in the resources section.

First the constructor of this class needs to register with OpenJMS for receiving messages in the queue defined for this Web Service. The constructor takes the Web Service name as an input argument. A static initialization block is used to instantiate the Axis engine and this gets executed when the Java Virtual Machine (JVM) loads the GenericJMSSOAPListenerForAxis. All the instances of this class have to use the same Axis engine instance.

```
...
public GenericJMSSOAPListenerForAxis(String webserviceName)
{
 ...
 this.webserviceName = webserviceName;
 ...
}
```

AxisJMSListenersStarter class's main method starts the listeners by reading the XML file "jmswebsvcs.xml" that contains the list of Axis Web Services, creating and starting one instance of GenericJMSSOAPListenerForAxis for each service in the list by passing the name of the Web Service as an argument to the constructor. In effect this dynamically creates the concrete service endpoint destinations for the JMS protocol, since each Web Service now gets a unique concrete address.

```
public static void main(String[] args)
{
 // Read the JMS Web Services names from the xml file.
 ...
 org.w3c.dom.Document servicesListDoc = db.parse(inFileFullPath);
 org.w3c.dom.NodeList servicesList = servicesListDoc.getDocumen-
tElement().getElementsByTagName("service");

 for (int k = 0; k < servicesList.getLength(); k++)
 {
  String webSvcName = ((org.w3c.dom.Element) servicesList.
item(k)).getFirstChild().getNodeValue();
  new GenericJMSSOAPListenerForAxis(webSvcName);
 }
 ...
```

Now let's look at the message-handling logic in the GenericJMSSOAPListenerForAxis that's instantiated for a particular Web Service, say, "MessageService." Its onMessage() method is called once the message arrives in the queue named "MessageService." After creating the Axis MessageContext for the message the onMessage method sets the serviceHandler field of the MessageContext to tell Axis that the Web Service being invoked is "MessageService" and that for executing service-specific functionality, the service implementation class, as defined in the server-config.wsdd, for service name "MessageService" should be invoked.

```
public void onMessage(javax.jms.Message inMsg)
{
 ...
 axisMsgCtxt.setRequestMessage(axisSoapMessage);
 // Set the target Web Service in the Axis message context to
indicate that this message should
 // go to the Axis webservice named <webserviceName> for which
this queue is receiving messages.
 axisMsgCtxt.setTargetService(webserviceName);
...
}
```

This method then sends the response SOAP message to the JMS client. However, at this point no correlation id is used. For simplicity's sake it's assumed here that the client, after sending the message, waits on a receive queue expecting to get a response message for the Web Service invocation it just made. This listener class can easily be extended to refer to and use a client-specified JMS correlation id to send a correlated response to the client.

Invoker-side implementation

Now let's look at some key aspects on the client side of the invocation. The classes written for the client side are JMSTestClientRPC, MyJMSTransportForAxis, JMSTestClientMessageStyle – which are client classes that invoke RPC- and message-style services respectively, for example they respectively invoke the services "urn:xmltoday-delayed-quotes" of samples.stock and "MessageService" of samples.message packages, the samples available in the Axis distribution – and MyJMSSender – which are the JMS transport handlers for the client side. JMSTestClientRPC makes a call using the Axis Call object to a given RPC Web Service using JMS as the transport. Hence it specifies the unique target endpoint JMS address defined for that particular Web Service, for example, here "urn:xmltoday-delayed-quotes." Please note that the RPC operation name is set with just the method name as given in the WSDL file and no service name prefix. The Axis Call object has to be told that a JMS transport handler needs to be used for this invocation. This handler class is instantiated and attached to the call here. If this value isn't set, Axis will use the HTTP transport handler by default. For both RPC- and message-style invocation, the same MyJMSTransportForAxis and MyJMSSender classes are used and connected via an entry made in the client-config.wsdd file available in the client classpath. MyJMSTransportForAxis class helps Axis locate this entry in the client-config.wsdd. MyJMSSender's job is to actually send the SOAP message as a JMS message to the Web Service queue destination specified here and get the response from the response queue.

```
...String webSvcJMSDestination = "urn:xmltoday-delayed-quotes";
axisCall.setProperty(org.apache.axis.transport.jms.JMSConstants.
DESTINATION, webSvcJMSDestination);
axisCall.setOperationName("getQuote");
...org.apache.axis.client.Transport transport = new
MyJMSTransportForAxis();
axisCall.setTransport(transport);
...
```

Running the Listeners

Install OpenJMS and Axis. Unzip the download.zip into a windows folder named, say, A. Modify the setclasspath.bat to give the correct value for Axis Home and openjms home. Put the server-config.wsdd in your Axis installation's (the axis zone of your web server's webapp) WEB-INF folder if there is none already. If you already have this file then copy the two service element entries in full to the already existing server-config.wsdd. Define three queues in openjms.xml, i.e., one for each Web Service given in jmswebsvcs.xml and one for "replyq," which is the reply queue used by the client. Change the GenericJMSSOAPListenerForAxis code line at the top to specify the full path of your Axis installation's server-config.wsdd file. Compile the source code to create .class files. Put the .class files in folder A with the right package structure. Now start openjms. Go to folder A in the DOS command prompt. Run setclasspath.bat. Then start the JMS listeners using the command java jms.AxisJMSListenersStarter.

Now, to run the test client, open another command prompt window. Go to folder A. Run setclasspath.bat. Use the command java invoker.jms.JMSTestClientRPC to run the JMS RPC client. Then use the command java invoker.jms.JMSTestClientMessageStyle to run the JMS message service client. Make sure that folder A contains the client-config.wsdd and that folder A is first in the class path order.

Summary

This article has introduced a working JMS listener for use with Apache Axis 1.x and has shown how clients can uniquely address Axis Web Services for invocation over JMS and how they invoke them over JMS using a combination of this listener and custom-written JMS transport handlers.

The source code for this solution can be downloaded from the online version of this article at http://java.sys-con.com.

Resources

- Axis: http://ws.apache.org/axis/java/index.html
- Openjms: http://openjms.sourceforge.net/
- Web Services Description Language (WSDL) 1.1 - March 15, 2001 http://www.w3.org/TR/2001/NOTE-wsdl-20010315
- JMS: http://java.sun.com/products/jms/
- Some articles on JMS with Axis:
  - "Programming JMS Applications using Axis" (IBM developerworks) http://www-128.ibm.com/developerworks/webservices/library/ws-jms/
  - "Axis meets MOM" (javaworld.com) http://www.javaworld.com/javaworld/jw-02-2006/jw-0220-axis.html

"Apache Axis is a popular Java-based Open Source **platform for exposing Web Services"**

AJAX for Java

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

The Backbase AJAX Java Edition:

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

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

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

Data Access Service

How to access relational data in terms of Service Data Objects

by Kevin Williams & Brent Daniel

Service Data Objects (SDOs) have become a foundation technology for Service Oriented Architecture (SOA). Recently, BEA, IBM, Oracle, SAP, Iona, Siebel, and Sybase announced their support for an SOA-enabling framework specification named Service Component Architecture (SCA). SD O provides the primary data representation in this framework.

Although not addressed by the current SDO or SCA specifications, there's a definite need for a generic data access service that operates in terms of SDOs. The alternative to this service would be the tedious and error-prone development of a custom mapping between the back-end data representation and Service Data Objects.

The Relational Database Data Access Service (RDB DAS) obviates the need for this custom development by providing a robust data access utility built around SDO. Because of its tight integration with SDO, the RDB DAS is also a perfect solution for data access in an SCA-based application.

By employing the RDB DAS, applications avoid the details and complications of working directly with a relational database and also the complex transformation between relational rows/columns and Data Object types/properties.

Background

Since the release of the specification in late 2003, SDO has proven itself a flexible and robust technology for data representation. Its inherent support for disconnected operations and heterogeneous data sources offers strong support for the needs of modern software architectures. For these reasons, SDO has found its way into several commercial products from major vendors and these same characteristics have led to its inclusion in SCA as a foundation technology.

SDO provides the general case mechanism for moving data around an SCA-enabled application. However, the reality is that most of this data must originate in some database at one edge of the application and be stored in some database at another edge. Unfortunately, database access isn't currently either SDO or SCA. (An early version SDO Data Access Service specification is in progress.)

This leaves the developer with a serious undertaking since there's a fundamental mismatch between the objects that an application works with and the tables and rows of a relational database that provide the persistent store for the object's state (see http://en.wikipedia.org/wiki/object-relational_impedance_mismatch).

For example, let's consider a simple query against a relational database for customers in a certain age range and their related orders.

An SDO-enabled application could most easily and naturally work with a normalized graph of Data Objects representing the query. Figure 1 illustrates this graph of connected Data Objects.

This in-memory graph of data objects brings to bear all of the capabilities of SDO.

- It's a disconnected representation of the queried data
- It provides simple traversal between related elements
- It tracks all changes from its original form via the SDO change summary
- It contains no redundant information
- It's easily serialized to XML

But unfortunately the relational database returns a tabular representation of the query result complete with redundant customer information as shown in Figure 2.

The transformation required to convert from tabular format to a graph of interconnected data objects is complicated and the reverse (transforming graph changes to a sequence of SQL inserts/updates and deletes) is even more so.

Because of the difficulties inherent in the transformation between the database and the application object space, an application development project can easily spend a third of its development resources on functions related to moving object state in and out of the database.

Business application developers shouldn't be burdened with this task and should instead be allowed to focus on business functionality.

Solution

The RDB DAS offers a solution to the problems mentioned above by providing two major capabilities. The RDB DAS can:

1. Execute SQL queries and return results as a graph of Data Objects
2. Reflect changes made to a graph of Data Objects back to the database

Figure 3 illustrates these two capabilities in a typical client interaction. The client starts by reading a graph of data specified by some query. The client then makes modifications to the graph, possibly by adding elements, and then requests the DAS to push the changes back to the database.

The DAS provides an intuitive interface and is designed so that simple tasks are simple to complete while

Kevin Williams is a software developer with IBM and is leading IBM's participation in the DAS subproject of the Apache Tuscany incubator.

Brent Daniel is a software developer working on SDO related technologies for IBM. He is a major contributor to the DAS subproject of the Apache Tuscany incubator.

more complicated tasks are just a little less simple.

The application interface to the DAS is based on the familiar Command Pattern and interaction with the DAS consists of acquiring command instances and executing them (see *Design Patterns* by Erich Gamma, et al). The following example demonstrates the simplest possible read of data.

```
Command read =
    Command.FACTORY.createCommand("select
* from CUSTOMER where ID = 10021");

read.setConnection(getConnection());
DataObject root = read.executeQuery();
```

In this case the command is created programmatically from a Command factory and the only input necessary is the SQL SELECT statement. Executing the read command returns the root of the resulting data graph and data can be extracted from the graph using the SDO dynamic API.

```
String lastName =  root.
  getString("CUSTOMER[1]/LASTNAME");
```

Pushing changes back to the database can be equally straightforward. Continuing with this example we can modify the customer object and then direct the DAS to send the modifications to the database. This line uses the SDO dynamic API to change the last name of the retrieved customer.

```
root.setString ("CUSTOMER[1]/LASTNAME",
"Williams");
```

Now that we have a modified graph, we can synchronize the changes with the database by passing the data graph to an "apply changes command" and asking it to execute.

```
ApplyChangesCommand apply = Command.
FACTORY.createApplyChangesCommand();
apply.setConnection(getConnection());
apply.execute(root);
```

As you may have noticed, the read and write examples each required three lines of code (except the code to get the connection object). So those of you familiar with O/R frameworks might be asking yourself a few questions. What is going on here? Where did you define all the configuration data? I didn't see a deployment descriptor? Where is the object-relational mapping information? Where are the static domain classes like Customer? The answers to these questions are based on two significant SDO capabilities and one design philosophy:

- Dynamic SDO
- SDO Change History
- DAS use of convention

Dynamic SDO

The reason you don't see a Customer interface or class used in this example is because the DAS can work with dynamic SDO data objects. This is a very powerful and often overlooked SDO capability.

Many applications today use the Transfer Object(TO) pattern to move data around tiers within an application (see *Core J2EE Patterns* by Deepak Alur, et al). Since these TOs typically have no behavior, there's little justification for Java interfaces and classes to implement the TO. These artifacts just represent more code to write, maintain, and manage.

One argument for TOs as Java interfaces/classes is the potentially cleaner API:

```
Static API
customer.setLastName("Williams")
Dynamic API
customer.setString("lastName", "Williams")
```

However, the SDO dynamic API is straightforward and can even be simpler to read than a static equivalent. For example, we can use the SDO XPath capability to access properties like this:

```
amount = customer.getFloat("orders[17]/
price");
```

The equivalent, with normal static Java APIs, would look something like this:

```
amount = ((Order)customer.getOrders().
get(17)).getPrice();
```

The dynamic API can also be useful in applications where the data model is likely to change often during development. It lets developers use the full breadth of Data Object function without having to generate a new static model (Java classes and interfaces) every time a change is made.

SDO Change History

The change history feature of SDO data graphs is another reason that SDO data objects can be thought of as transfer objects on steroids. Not

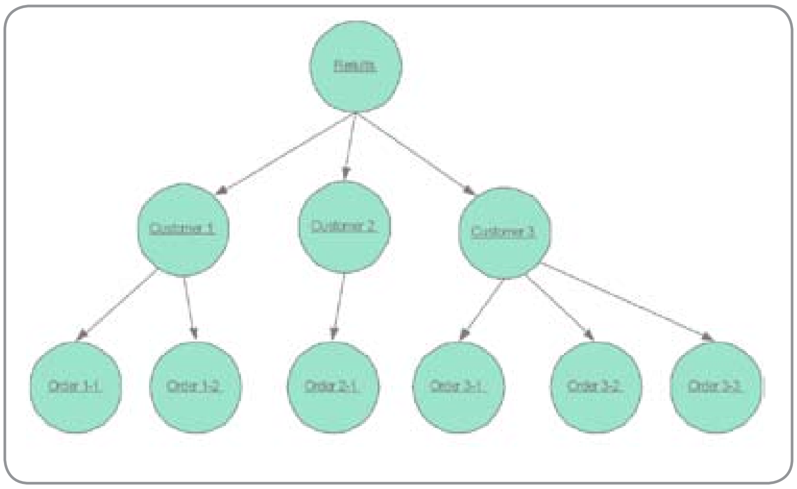
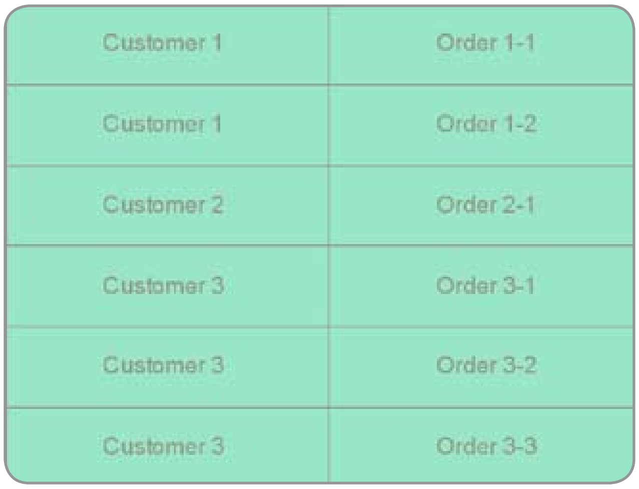


Figure 1 DMA operation by abstraction and selection



Customer 1	Order 1-1
Customer 1	Order 1-2
Customer 2	Order 2-1
Customer 3	Order 3-1
Customer 3	Order 3-2
Customer 3	Order 3-3

Figure 2 DMA operation by abstraction and selection

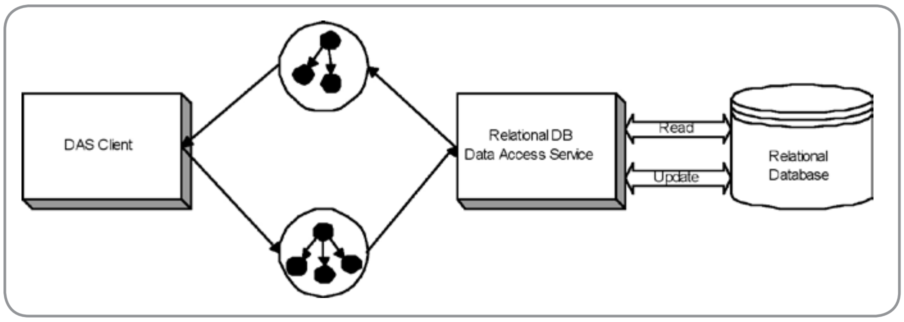


Figure 3 DMA operation by abstraction and selection

only do data objects provide a snappy dynamic API and XML serialization, SDO data objects also remember any changes that have been made to them.

The change history capability means that SDO data objects aren't dependent on a container or some persistence manager to track their state. In fact, since the change history is serialized along with the associated data objects, a graph of SDO data objects can flow through different tiers of a distributed application remembering all the changes that may occur along the way. Later, when it's time to reflect those changes back to the database, the DAS can process the change history and build the set of create/update/delete commands needed to flush the accumulated changes.

The Change History tracks changes made to all data object properties including fields and relationships. Using this information, the DAS can handle the complex task of reflecting object graph changes back to the database without exposing this complexity to users. The DAS translates object property changes into database column updates and object relationship changes into database foreign key updates.

Use of Convention over Configuration

The DAS makes use of convention to simplify the programming model. For instance, in the simple read example above we have this statement to access the last name of a customer:

```
String lastName =  root.
getString("CUSTOMER[1]/LASTNAME");
```

Notice the path name: "CUSTOMER[1]/LASTNAME". This suggests that there is an SDO Type named CUSTOMER with a property named LASTNAME. If you remember, the command used to read this data was created like this:

```
Command read =
    Command.FACTORY.createCommand("select *
from CUSTOMER where ID = 10021");
```

The RDB DAS, by convention, creates an SDO Type for each database table represented in the query result. In addition, it creates a property for each table column represented in the query result. In the absence of any additional configuration data, the names of these Types and Properties will exactly match the names of the database Tables and Columns. So given the SELECT statement above and the knowledge that the CUSTOMER table has a column named LASTNAME, we can assume that the data graph returned will be populated with instances of Type CUSTOMER that have a property LASTNAME. This capability is made possible by using the metadata associated with the ResultSet returned from the query execution.

If the application developer wants the names of Types and Properties to vary from the names of the Tables and Columns then he or she can override this convention with a bit of configuration. We'll get into the details of providing configuration to the DAS a little later.

Another bit of convention that this example demonstrates is exploited when flushing graph changes to the database:

```
ApplyChangesCommand apply = Command.
FACTORY.createApplyChangesCommand();
apply.setConnection(getConnection());
apply.execute(root);
```

In the absence of instruction (configuration) to do otherwise, the DAS will scan the change history and generate the create/update/delete (CUD) statements necessary to flush the changes to the database. Since we just changed a single property of a single data object, the change history processing produces a single statement to be executed:

```
update CUSTOMER set LASTNAME = 'Williams'
where ID = 10021
```

There are a couple of things we'd like to point out here. The first one has nothing to do with convention but it's very cool. What has been generated here is a "partial update." That is, rather than generating a complete update statement that covers every column in the table, the statement only updates columns that relate to changed data object properties (i.e., just the last name).

Partial updates may not be the right way to go for some applications so CUD generation can be overridden with user-supplied CUD statements. However, partial update is a good fit for many applications and with it you can avoid a great deal of configuration or additional programming. Not only that, partial updates provide a performance boost for updates to tables with very wide rows and are also useful for avoiding database triggers.

The other point we want to make has to do with the "where" clause ("where ID =") of the generated update statement. Since we mean to update the specific table row that's associated with the modified data object, we need to qualify the update statement with a unique row identifier. So this is where another piece of convention is used. If the DAS isn't provided with configuration that defines a unique identifier for the data object Type then the DAS will look for one. There's no magic here; if there's a property named ID then the DAS will assume it's unique and use it in the "where" clause.

We've provided a description of the convention currently employed by the DAS. But there's more on the way. We're currently looking to add more capability based on conventions for generated columns, optimistic concurrency control, and relationship definition.

#1 Rated AJAX and Rich Internet Application toolkit* TIBCO General Interface

Build 100% Pure Browser Rich Internet Apps with TIBCO General Interface™

"TIBCO General Interface is a friendly, capable toolkit for building sophisticated JavaScript Web applications that run in a browser."

InfoWorld 2006 TECHNOLOGY OF THE YEAR AWARD

Download today
at **http://www.tibco.com/mk/gi**

TIBCO®
The Power of Now®

The use of convention isn't revolutionary or even new, but it is gaining renewed respect. This may be a reaction to the configuration-heavy frameworks we've been using in recent years. Notably, Ruby on Rails, Maven, JUnit, Wiki and many other "agile" frameworks make considerable use of convention over configuration. It's amazing what can be done easily, and how much coding and configuration can be avoided with these tools by adhering to simple conventions.

We've explained how the DAS leverages the capabilities of SDO and makes use of convention to provide a progressive programming model. Now we'll walk through a complete example that demonstrates a few more RDB DAS capabilities.

A Complete Example (CompanyWeb)

In this example we'll display the steps involved in writing a simple application to work with companies and their related departments. First we need to introduce a new DAS concept; the DAS Configuration model. Although we're adding more options for leveraging conventions, there are still capabilities in the DAS that require configuration such as relationship definitions and database-generated IDs.

The DAS Configuration can be built up programmatically or loaded via an XML file. In this example we'll use the XML file approach.

We'll begin by accessing data from the Company table, defined as follows:

```
COMPANY:
ID  NAME
```

We start by creating an XML file and add descriptive information for the database tables and columns. The snippet of XML below tells the DAS that the COMPANY table has a primary key column named ID that is auto-generated by the database:

```
<Table name="COMPANY">
       <Column name="ID" primaryKey="true"
generated="true"/>
 </Table>
```

Notice that we do not define the NAME column. There's nothing special about this column so we'll just take the conventional behavior offered by the DAS.

In the earlier examples we had the client pass a connection instance to the DAS for use during execution. An alternative is to define connection properties in the Config and have the DAS manage the connection for us. Here we choose to use a DataSource and provide the JNDI name:

```
<ConnectionProperties dataSource="java:
comp/env/jdbc/dastest"/>
```

Finally, we'll define a Command that the DAS will use to access the data. The following command will retrieve all companies from the database:

```
 <Command name="all companies" SQL="select
* from COMPANY" kind="Select"/>
```

Now we can write an application to access the data and create a class called CompanyClient to handle interaction with the DAS. However, first we'll introduce a new DAS concept: the CommandGroup.

A CommandGroup is a logical grouping of commands and associated configuration data that serves two main purposes. Applications will often define commands that require the same configuration information and a CommandGroup binds the defined commands and the provided configuration data. For example, commands in the same CommandGroup will share the same connection properties and relationship definitions.

Secondly, a CommandGroup is initialized with Commands that it provides by name. Since the client retrieves commands by name and then executes them, the SQL-specific configuration can be contained in the group and isolated from the application. In theory, the same application could switch to using some other data store technology by changing the way the Config is initialized. For example, a Config could be initialized to use static SQL or even a non-relational back-end.

Since our application will use commands that share configuration, we'll use a CommandGroup and create one CommandGroup instance in CompanyClient and initialize it with our XML file.

```
private CommandGroup commandGroup =
        CommandGroup.FACTORY.createCommand
Group(getConfig("CompanyConfig.xml"));

private InputStream getConfig(String file-
Name) {
        return getClass().getClassLoad-
er().getResourceAsStream(fileName);
   }
```

Now we'll create a method to return a List of Company DataObjects:

```
 public List getCompanies() {

        Command read = commandGroup.
getCommand("all companies");
        DataObject root = read.execute-
Query();
        return root.getList("COMPANY");
     }
```

At this point, we have an application capable of returning a list of all companies in the database. Now let's add in another database table, Department:

```
DEPARTMENT:
ID NAME LOCATION        NUMBER      COMPANYID
```

The Department table is also using a primary key named "ID" that is auto-generated by the database, so its table definition will be similar to that of Company:

```
<Table name="DEPARTMENT">
 <Column name="ID" primaryKey="true"
generated="true"/>
</Table>
```

We have to define the relationship between Company and Department so that the DAS can construct a dynamic SDO model with a relationship between the two and correctly maintain those relationships in the database. The following XML snippet names the relationship, associates the keys, and specifies the cardinality:

```
<Relationship name="departments"
    primaryKeyTable="COMPANY"
    foreignKeyTable="DEPARTMENT"
many="true">
   <KeyPair primaryKeyColumn="ID" foreignKe
yColumn="COMPANYID"/>
</Relationship>
```

Now we can add a command to return all companies and departments:

```
<Command
```

```
   name="all companies and departments"
   SQL="select * from COMPANY left outer
join DEPARTMENT on COMPANY.ID =
       DEPARTMENT.COMPANYID"
    kind="Select"/>
```

Next we add a method to CompanyClient to access and execute this command. This method returns a list of Company data objects, but since the command employs a join with Departments, each Company will have its related Department data objects associated with it.

```
  public final List getCompaniesWithDepart-
ments() {
        Command read = commandGroup.
getCommand("all companies and depart-
ments");
        DataObject root = read.execute-
Query();
        return root.getList("COMPANY");
    }
```

Next we'll add the ability to retrieve a single company and all its departments. The configuration file is updated with this command definition:

```
<Command name="all departments for company"
   SQL="select * from COMPANY left join
DEPARTMENT on COMPANY.ID =
    DEPARTMENT.COMPANYID where COMPANY.ID
= :ID" kind="Select"/>
```

Note that we have defined a named parameter ":ID" in the SQL query. The CompanyClient uses the code below to access this command:

```
public final List getDepartmentsForCompany
(int id) {
        Command read = commandGroup.
getCommand("all departments for company");
        read.setParameterValue("ID", new
Integer(id));
        DataObject root = read.execute-
Query();
        return root.getList("COMPANY[1]\
departments");
    }
```

Now we'll add a write capability to CompanyClient. Since we'll let the DAS generate the CUD statements, no additions are necessary to the configuration file.

```
public final void addDepartmentToFirstCom-
pany() {

        Command read = commandGroup.
getCommand("all companies and depart-
ments");
        DataObject root = read.execute-
Query();
        DataObject firstCustomer = root.
getDataObject("COMPANY[1]");

        DataObject newDepartment = root.
createDataObject("DEPARTMENT");
        newDepartment.setString("NAME",
"Default Name");
        firstCustomer.
getList("departments").add(newDepartment);

        ApplyChangesCommand apply = com-
mandGroup.getApplyChangesCommand();
        apply.execute(root);

    }
```

A complete example based on this company and department scenario, including a Web application used to access the CompanyClient, is available at the Apache Tuscany incubator project. The readme is available at http://incubator.apache.org/tuscany/samples/java/samples/das/company-web/readme.htm.

The complete source is here: http://svn.apache.org/repos/asf/incubator/tuscany/java/samples/das/company-web/

In the space of this article we've shown some of the main capabilities of the Relational Database Data Access Service being developed at Apache's Tuscany incubator project. Here are other important supported capabilities:

- Statically typed (generated) SDO DataObjects
- Optimistic concurrency control
- Stored procedures
- External transaction participation
- Write-operation ordering (database constraints)
- Simple name mapping (Table/Column -> SDO Type/property)
- Column-type conversions
- Paging

Business Benefits

Object-to-Relational Data Access – The RDB DAS provides a capable and flexible data access mechanism to applications integrating SDO technology. By employing the DAS, developers avoid developing a custom data access framework, a task that's tedious, complex, and error-prone.

Integrated with SDO – The Transfer Object pattern is often used by applications to move persistent state from one part of the application architecture to another. This is especially true if the data movement requires serialization. Such an application can employ some object-to-relational technology (JDO, EJB, Entity beans, etc.) to retrieve the data from a back-end data store and then copy the data to the DTO for transfer around the application.

The creation of separate TOs isn't necessary for an SDO-integrated application using the DAS because the SDOs themselves are easily serialized to XML. As a bonus to the TO pattern, the SDOs "remember" changes made to them and this memory is preserved through serialization/de-serialization.

Conclusion

The RDB DAS and SDO provide a simple and powerful way to access and work with relational data. The RDB DAS lets developers work with SDO without building custom data access solutions since the DAS works in terms of SDOs. It simplifies data access by hiding many of its complexities while still letting developers harness more powerful features in complex scenarios.

Because the RDB DAS integrates SDO technology, it's a natural fit for data access in the SCA framework. In fact, an RDB DAS implementation is evolving as part of the "Tuscany" SOA Apache incubator project along with implementations of SCA and SDO. The DAS is also on the roadmap for the upcoming SDO 3.0 specification.

The examples and code included in this article can be had from the Apache Software Foundation and licensed according to the terms of the 2.0 Apache License.

More information about the RDB DAS and the implementation under development can be found at http://incubator.apache.org/projects/tuscany.

Building Real-Time Applications with Continuous Query Technology

The promise of a robust new development model

by Gideon Low & Jags Ramnarayan

The client/server development model prevalent in the mid-1990's resulted in extremely easy-to-build rich GUI applications that interacted directly with a relational database. 4GL tools such as Visual Basic and PowerBuilder let even junior developers visually compose both the presentation and most of the backend data binding. While this made for impressive Rapid Application Development (RAD) productivity, the client/server architecture was severely challenged when dealing with real-time environments where the data changes rapidly and applications require visibility to the correct data at all times. As a result, client applications were forced to poll the database continuously to check for changes.

The same is true in today's browser-based or Java Swing-based multi-tier applications, where the user is forced to issue a screen refresh to view the latest state. Real-time applications such as a trader desktop where the screens are continuously refreshed are still sophisticated proprietary applications that require specialized application design to push events from backend servers to the GUI clients. Such applications result in hundreds or even thousands of views like this: Maintain a continuous view of all Intel and Dell orders placed today and notify me when AMD moves up or down by 5%.

However, today a new "push-based" architecture enables data changes to be monitored continuously in a backend data management system and changes continuously pushed to client applications, maintaining a real-time view at all times.

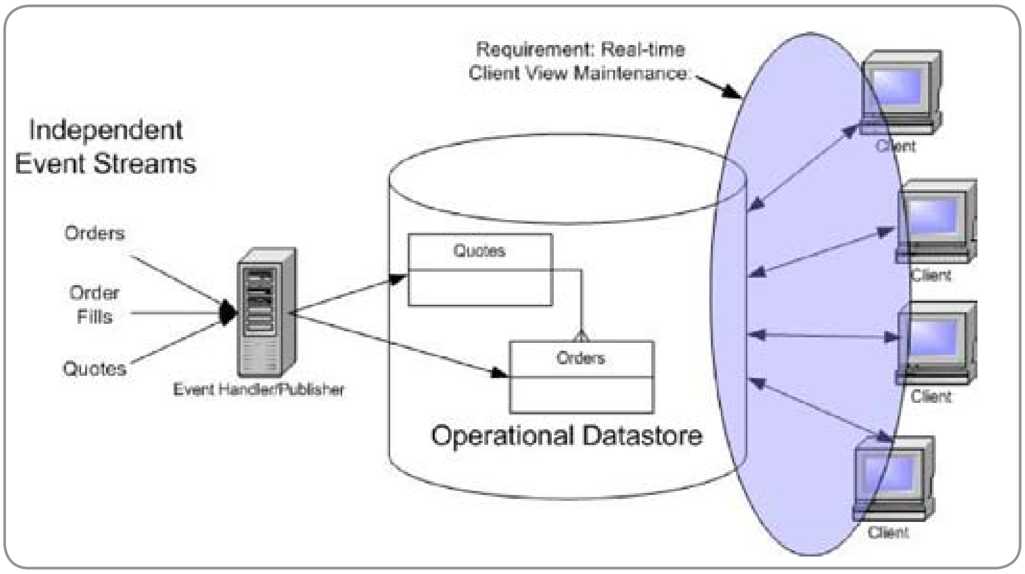


Figure 1

Traditional Databases Are Passive

Most complex GUI screens use complex SQL – multi-table joins, column aggregations, and multiple predicates for filtering, grouping, etc. to construct the dataset being viewed. Consider a real-time application like a financial stock monitoring program or a traffic management system with hundreds of concurrent clients with equally large numbers of complex queries that are continuously being executed once every second. The traditional relational database that's built for storing data efficiently and guaranteeing consistency won't be able to cope with this demand. Relational databases are passive, executing queries on sitting data only. Today's complex applications, however, require a system that can very efficiently execute queries as data streams in.

A SQL Continuous Query Engine – An Active Data Management System

By building a database engine designed specifically so queries can remain standing and active – or continuous — the scalability, reactivity, and organization of multi-tiered data-centric applications can be radically altered. Continuous queries (CQ) let users get new results from a database without having to issue the same query repeatedly.

Queries no longer have to be reissued to refresh result sets, logic that has to execute in response to complex changes in a data model can actively register interest directly from the source, and business logic can be safely co-located with application data in a relational model without scalability limitations.

Continuous querying technology works through an engine that efficiently groups and filters predicates from large numbers of queries, enabling several key things to happen:

- When the server first gets a continuous query, it not only replies with an initial result set, but it analyzes the query predicates (selection criteria) to group it logically with other similar queries.
- The engine can then quickly identify what continuous queries are affected by any given data modification (an insert, update, or delete against the relationally structured operational data).
- The engine can send only the deltas to each CQ client needed to update its existing result set, in effect exactly the data necessary for the client to hold a materialized view of data from the server.

The inherent power of this technology lies in both its simplicity and natural scalability. With an in-memory database and some very simple extensions to existing query languages, we're suddenly capable of building a middle-tier that combines all of a database's operational benefits and none of its limitations. At the same time, we can build application server clients that express interest in data through conventional queries without having to trade performance for data currency or functional sophistication for development effort.

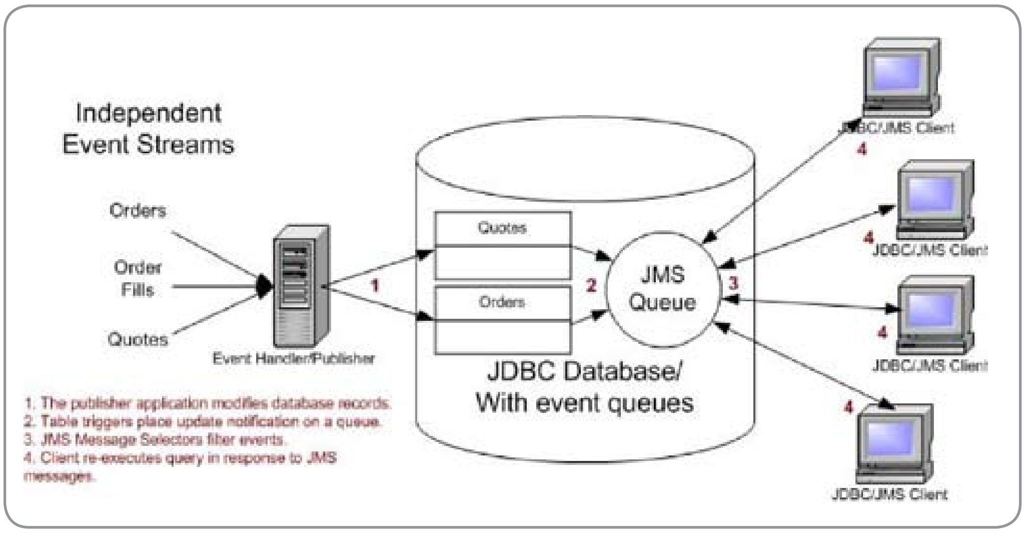
Where Does It Apply?

While the focus of this article is to illustrate the power of CQ to provide real-time view maintenance in graphical user interfaces, the power of this paradigm is well beyond this. Continuous querying is part of a new data management paradigm called stream data processing (see the References section below for further information) and can be used to monitor multiple streaming sources of data, analyze these streams for patterns of interest, and respond instantaneously. The sources of the data could be disparate – RFID sensor events, events from business applications across an enterprise, external sources, etc.

If the applicability of the technology were to be characterized in two points they would be:

1. Data is changing very rapidly and decisions have to be

Gideon Low is a senior technical architect at GemStone Systems. Gideon has over 10 years of experience in the development, management, and sales of high-performance large-scale distributed systems. His focus over the last seven years has been in high-speed electronic trading systems as CTO at Silicon Summit Technologies (A FIX/OMS vendor) and VP, client connectivity technology at Lehman Brothers. Gideon joined GemStone in 2005 as a senior architect to help bring GemStone's GemFire product to Wall Street's most demanding high-performance environments.



1. The publisher application modifies database records.
2. Table triggers place update notification on a queue.
3. JMS Message Selectors filter events.
4. Client re-executes query in response to JMS messages.

Figure 2

made instantaneously.
2. The system can analyze hundreds to thousands of patterns (rules or query predicates) with thousands of events pouring in every second.

Leveraging CQ in a Financial Order Tracking System

A great example of a system with stringent real-time data requirements is a securities trading order-tracking system. Diverse event streams such as customer orders, order executions, and market data quotes must be combined into a continuously updated view provided to multiple end users. Early implementations — built at a time of much smaller order volumes and slower workflows – used client/server architectures that required GUIs to poll the database for updates.

As trading volumes grew and firms realized that they could gain a competitive advantage with faster trading systems, we started seeing trading application servers that could publish real-time updates to trader GUIs. Implementing this functionality efficiently — with predictable low-latency and high throughput – required a much more sophisticated development model than client/server could provide and so the effort to build, maintain, and operate these systems grew quickly. The example in Figure 1 strips functionality to its bare essentials to illustrate the relative merits of conventional and CQ architectures.

Data Model

For simplicity's sake, our example uses only two tables — Orders and Quotes, as shown in Table 1.

The Orders table simply tracks the orders clients have sent you and the number of shares filled in each one. The Quotes table keeps track of market activity for any symbol that exists in the Orders table, so the GUI displays market activity associated with the order to the responsible broker. This means that we have three event streams coming into the system: orders from clients, order fills from a stock exchange, and market data quotes from a quote feed. These three event streams must be coalesced into a coherent view within the Order Tracking System. Naturally, in the real world these systems are much more sophisticated — requiring transactions, many more data entities and streams, and complex user interactions, but we'll keep things simple and explain later how the example may be extrapolated to additional complexity.

We'll describe the components and options for this use case in three sections. The first shows how to build a simple database publisher application for quotes and orders. The second shows how you might build a client application using plain JDBC, or with the addition of database and JMS queues, and third how you'd build a client application with Continuous Querying. Note that the full source code and pre-configured runtime configurations for the DataPublisher CQExampleClient applications are available for download at http://www.gemstone.com/download/.

A Simple Data Publisher for an Order Tracking System

The Data Publisher application is quite straightforward. It instantiates Order and Quote generation simulators and handles their events by "publishing" them in a JDBC database with SQL Insert and Update statements.

Let's take a closer look at the Publisher application by inspecting some of its code. The classes we need to understand are SimpleDataPublisher, SimpleQuoteGenerator, and SimpleOrderGenerator.

- SimpleOrderGenerator has a pre-defined list of Orders and Customers. Its constructor has only one argument — a listener that implements the methods onNewOrder(SimpleOrder o) and onOrderUpdate(SimpleOrder o). In the constructor it creates a basket of 63 orders (which results in onNewOrder() callbacks), and then spawns a thread to gradually fill the orders by incrementing the fillQty property of each one.

Jags Ramnarayan is the chief architect for the high-performance, distributed data management product line at GemStone Systems. He puts on multiple hats - evangelizing the technology, exploring requirements with customers, and managing the overall direction of the architecture decisions. In the past Jags participated in several Java and W3C standards for GemStone and BEA. On the side, Jags is also pursuing a MBA degree, but hopes to remain technically focussed.

- SimpleQuoteGenerator generates simple market data quote streams in response to calls to its addSymbol (String symbol) method. It's constructed with a listener that implements onQuote (SimpleQuote q), and it spawns a thread that randomly updates each Symbol's quote at a steady rate.

SimpleDataPublisher connects to a JDBC database with a JDBC driver, either creates or clears out the Quote and Order tables with DDL/DML statements, and then uses the two generator classes as event sources. Each generator callback event (onQuote(), onNewOrder() and onOrderUpdate()) fires logic to perform a SQL INSERT or UPDATE to the CQ server, thus creating or modifying records in the Quote and Order tables. It contains only the three following members:

```
Connection c = null;  // The JDBC Connection to the CQ Server
SimpleQuoteGenerator quoteGenerator = new SimpleQuoteGenerator (
this );
SimpleOrderGenerator orderGenerator = new SimpleOrderGenerator (
this );
```

The constructor of SimpleDataPublisher looks like Listing 2.

Using a handy helper class to hide the boring details, the code gets a handle on a valid JDBC Connection object, creates or clears the necessary tables, initializes the OrderGenerator (which causes an onNewOrder() callback for each new order), and then starts the QuoteGenerator and OrderGenerator threads to initiate the event streams.

Note that for each new order created, SimpleDataPublisher calls the SimpeQuoteGenerator.addSymbol() method to register a new Symbol for quote generation — thus making sure that we get quotes for each symbol handled by system (duplicate symbols are ignored).

Each time SimpleDataPublisher gets a callback invocation, it creates a SQL statement and submits it to the CQ Engine. For example, when onUpdatedOrder() fires, the logic in Listing 1 executes:

You can see that this is a very simple O/R mapping exercise plus JDBC database interaction logic. The logic that executes in response to onQuote() and onNewOrder() is fundamentally the same — and uses SQL/JDBC to update or insert a table record. Once started, SimpleExamplePublisher continues to update the order and quote records in response to the callbacks it's getting from the generators. This is a good time to reflect on the how this fits with your existing experience — the type of code above already exists in most systems, but how often does it serve as BOTH a database update and a notification to systems that have previously queried the updated record?

Now that we have an application that can publish our order and quote data to a JDBC database, let's explore different ways of building the client (or subscriber) application. We'll describe several approaches in varying levels of detail: JDBC polling, JDBC with simple JMS-based change notifications, and JDBC with continuous querying. To display data in a Swing GUI, we'll use a very handy component called the QuickTable (see http://quicktable.org for more info), which can build a very nice JTable from nothing more than a JDBC ResultSet.

Maintaining Real-Time Views Using Continuous Polling

A simple and somewhat naïve implementation would let the user set a refresh interval and continuously re-execute the query to refresh its view. The code in Listing 3 illustrates the JTable and the use of a thread to continuously refresh the view.

While simple, this method has several obvious disadvantages:

(1) With many concurrent views across hundreds of trader desktops, the database will be inundated with SQL select requests. In a more realistic application the queries will be more complex and the volume of data much larger, causing the database to buckle under pressure.

(2) It's quite inefficient to continuously re-execute the query, particularly when the underlying data hasn't changed. This is especially true when working with obscure stocks that may change only a few times during the trading day.

View Maintenance Using JMS

The most common approach in use today for real-time information management is to employ messaging to capture and route events from the backend database. Rather than route every single event, active filtering can be done in the backend servers through the use of JMS selectors. More information on JMS and JMS selectors can be found at http://java.sun.com/products/jms/.

Modern databases all support the asynchronous capture and propagation of database events. For instance, with Oracle, a simple mechanism would be to use a row-level trigger on the order and quote table and route the DML events to an Oracle Advanced Queue (AQ). The AQ can then be accessed

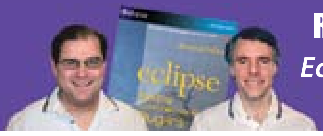
Table Orders:

Column Name	Data Type	Comment
OrderID	Varchar	Unique Identifier for an Order and the Primary Key of this table.
Symbol	Varchar	What you want to order (e.g., IBM, MSFT, etc.).
Price	Numeric	Price you want to buy or sell (only used if OrderType is the "Limit").
Quantity	Integer	How many shares to buy or sell.
Side	Char(4)	Either "Buy" or "Sell."
OrderType	Char(6)	Either "Market" or "Limit."
FilledQty	Integer	How many shares have been filled on the order so far. When the order is completely filled, this equals Quantity.
Client	Varchar	The name of the Client who sent you this order.
UpdateTimestamp	DateTime	Timestamp of the last update to row.

Table Quotes:

Column Name	Data Type	Comment
Symbol	Varchar	For example, IBM, MSFT, etc. Also Primary Key.
Bid	Numeric	Current price to sell.
Ask	Numeric	Current price to buy.
BidSize	Integer	Number of shares available at Price.
AskSize	Integer	Number of shares available at Price.
LastQty	Numeric	Quantity of the last executed trade.
LastPrice	Numeric	Price of the last executed trade.
UpdateTimestamp	DateTime	Timestamp of the last up

Table 1



RCP Developer™ is created by the experts who brought you the popular book
Eclipse: Building Commercial Quality Plugins — Eric Clayberg & Dan Rubel

Java on the desktop is back!

RCP Developer™

eclipse READY™

Instantiations RCP Developer harnesses the power of the Eclipse Rich Client Platform. Quickly create native desktop applications that are rich, cross-platform, and high-performance. The vast expertise of the Eclipse developer community and the experience of Instantiations lets you...

STAND ON THE SHOULDERS OF GIANTS

Create blazing fast rich Java clients.

Tap into the ultra high performance Eclipse graphical user interface libraries and ensure a native look and feel.

Build rich Java client applications faster.

Focus on creating business specific functionality rather than reinventing the low-level logic required for applications to run.

Deliver a more consistent user experience.

Tame the most demanding user interface requirements and reliably deliver a rich and consistent user experience across all applications.

Fully exploit the Eclipse Rich Client Platform.

"The Eclipse Rich Client Platform is the leading framework for creating Java client applications. RCP Developer is the first product to bring comprehensive application construction, GUI testing and packaging of rich-client applications to Eclipse RCP."

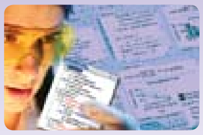
—Mike Milinkovich, executive director of the Eclipse foundation

RCP Developer™

SWT Designer™

Design. Quickly create views, editors, perspectives, and preference pages for your applications. Intuitive visual design and access to the high performance SWT GUI framework of Eclipse RCP.

WindowTester™



Test. Automatically test the GUIs that comprise Eclipse RCP applications. Minimize the need to hand-code GUI test cases. Automated recording, test generation, assertion coding and playback facilities.

RCP Packager™

Deploy. Streamline build and deployment for Eclipse RCP applications. Assemble application elements into a single package. Quickly create and maintain high-quality Eclipse RCP installations.

Download a risk-free trial copy:
www.instantiations.com/rcpdeveloper

instantiations
the development tools experts

www.instantiations.com 1-800-808-3737

The leader in Eclipse RCP development tools

eclipse FOUNDATION MEMBER

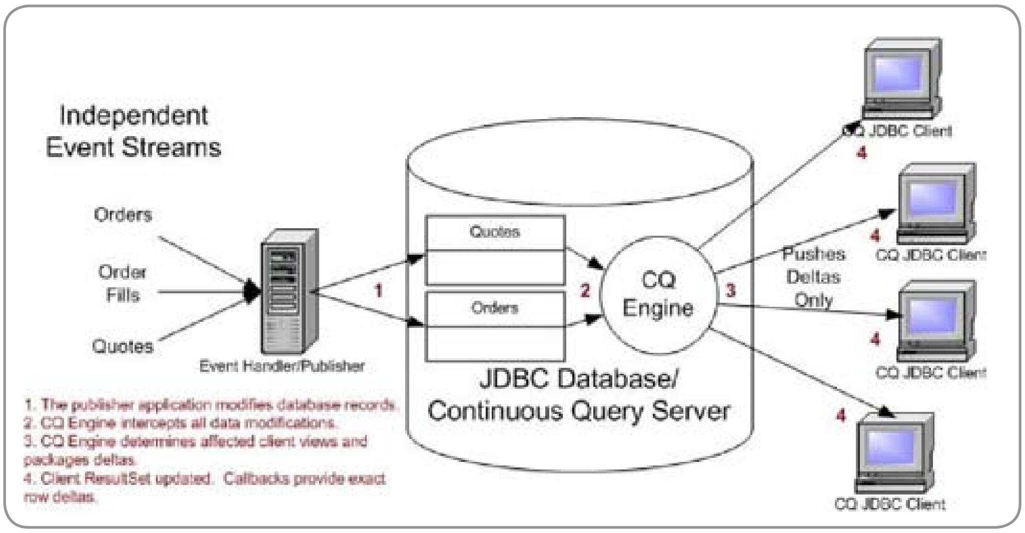

Independent Event Streams
Orders
Order Fills
Quotes
Event Handler/Publisher
1
Quotes
Orders
2
CQ Engine
3
Pushes Deltas Only
4
CQ JDBC Client
JDBC Database/ Continuous Query Server
1. The publisher application modifies database records.
2. CQ Engine intercepts all data modifications.
3. CQ Engine determines affected client views and packages deltas.
4. Client ResultSet updated. Callbacks provide exact row deltas.

Figure 3

from remote machines as a JMS destination (Queue).

The code in Listing 4 illustrates how our example in Figure 2 can be changed to make use of JMS events to refresh the JTable. Though this method doesn't require the application client to continuously execute queries, it still requires the client to re-execute the complete query to the database. One could argue that the database event generator could provide sufficient information in the message so as to reconstruct the client JTable from the message itself. But, in most real-life situations, the query could be quite complex involving joins, complex query predicates, projections involving aggregate column values and such, making it nearly impossible to calculate how exactly the client-side view has been impacted.

Again, much like the first naïve approach, with many concurrent clients with hundreds or even thousands of views to maintain, the approach quickly stretches the limits of database scalability.

Real-Time View Maintenance with Continuous Query

Now that we have examined polling- and messaging-based approaches to building a client application for Order Status Tracking, let's take a look at how we might build a better mousetrap with Continuous Querying (See Figure 3). To build this example, we use the GemFire Real-Time Events (RTE) Server — a JDBC in-memory database product that provides a robust CQ Engine.

The first requirement is to re-configure the DataPublisher application to connect to the RTE CQ Server JDBC database instead of a traditional RDBMS. Thanks to standard JDBC, this only means changing the JDBC driver classname and connection URL parameter. Nothing else in the DataPublisher applications needs to change as the database interaction is all in standard SQL-92, which all JDBC databases support.

The client side is encapsulated in the CQExampleGUI class, which, again, requires very little code beyond standard JDBC. Less, in fact, than the polling- and JMS-based client examples described above. The main difference is the addition of some logic to handle Continuous Query callbacks instead of a JMS listener or polling thread. All of the logic is encapsulated in the CQExampleGUI class:

For this simple example, CQExampleClient has only three important members:

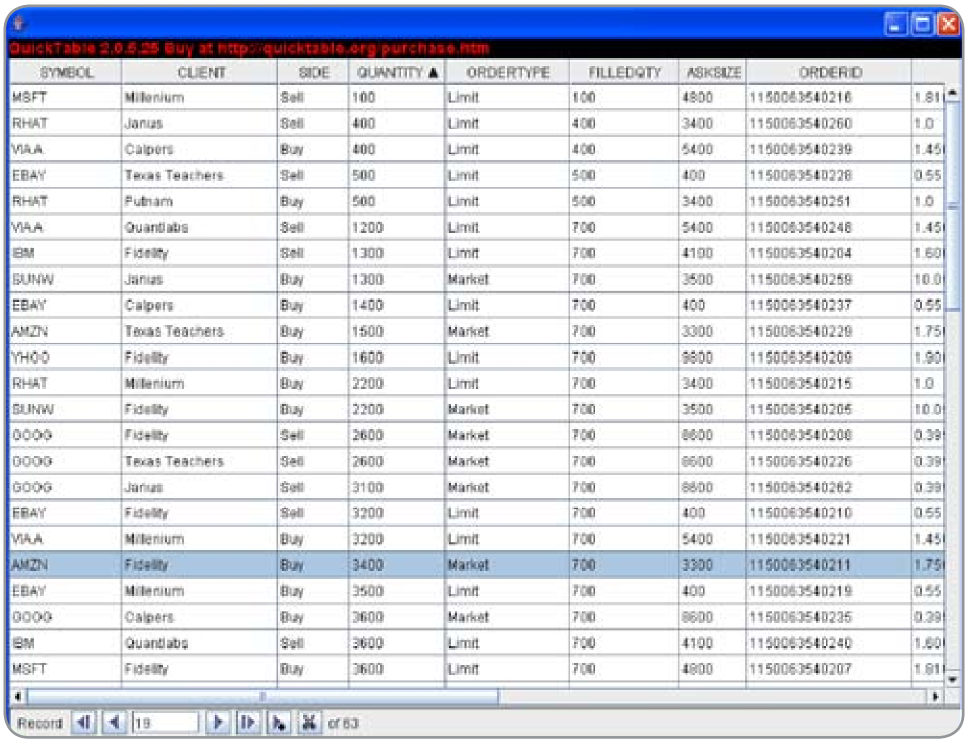
```
Connection c = null; // The JDBC Connection to the CQ Server
CQManager cqManager;  // The Continuous Query manager
DBTable dBTable1 = null; // A handy QuickTable for rich visual
display
```

The constructor for CQExampleGUI looks like Listing 5. With these few lines of code, we've painted a nice GUI with the contents of our simple order management database. Note that though the SQL Select statement used to initiate the Continuous Query is relatively simple, most Financial Order Tracking systems actually have a "view-builder" screen that lets users select columns and filter conditions, which results in much more complex dynamically generated SQL.

Now that we've built a GUI in much the same way as traditional client/server, what remains is to handle the CQ update events. The logic for this is implemented in the afterResultsUpdated ( ) method, which passes in an CQUpdate argument containing a collection of RowDelta objects and a handle on the refreshed JDBC ResultSet. With the convenience of a QuickTable, only one line of code is necessary to update the GUI:

```
dbTable1.refresh (cqUpdate.getResultSet(); );
```

In a more sophisticated application, you can handle the update in granular detail by iterating over the provided RowDeltas, which give you the update type (UPDATE, INSERT, or DELETE), an array representing the old row column values, an array representing the new row column values, and a list of the columns actually modified. In short,


QuickTable 2.0.5.25 Buy at http://quicktable.org/purchase.htm

SYMBOL	CLIENT	SIDE	QUANTITY ▲	ORDERTYPE	FILLEDQTY	ASKSIZE	ORDERID	
MSFT	Millenium	Sell	100	Limit	100	4800	1150063540216	1.81
RHAT	Janus	Sell	400	Limit	400	3400	1150063540260	1.0
VIAA	Calpers	Buy	400	Limit	400	5400	1150063540239	1.45
EBAY	Texas Teachers	Sell	500	Limit	500	400	1150063540228	0.55
RHAT	Putnam	Buy	500	Limit	500	3400	1150063540251	1.0
VIAA	Quantlabs	Sell	1200	Limit	700	5400	1150063540248	1.45
IBM	Fidelity	Sell	1300	Limit	700	4100	1150063540204	1.60
SUNW	Janus	Buy	1300	Market	700	3500	1150063540259	10.0
EBAY	Calpers	Buy	1400	Limit	700	400	1150063540237	0.55
AMZN	Texas Teachers	Buy	1500	Market	700	3300	1150063540229	1.75
YHOO	Fidelity	Buy	1600	Limit	700	9800	1150063540209	1.90
RHAT	Millenium	Buy	2200	Limit	700	3400	1150063540215	1.0
SUNW	Fidelity	Buy	2200	Market	700	3500	1150063540205	10.0
GOOG	Fidelity	Sell	2600	Market	700	8600	1150063540208	0.39
GOOG	Texas Teachers	Sell	2600	Market	700	8600	1150063540226	0.39
GOOG	Janus	Sell	3100	Market	700	8800	1150063540262	0.39
EBAY	Fidelity	Sell	3200	Limit	700	400	1150063540210	0.55
VIAA	Millenium	Buy	3200	Limit	700	5400	1150063540221	1.45
AMZN	Fidelity	Buy	3400	Market	700	3300	1150063540211	1.75
EBAY	Millenium	Buy	3500	Limit	700	400	1150063540219	0.55
GOOG	Calpers	Buy	3600	Market	700	8600	1150063540235	0.39
IBM	Quantlabs	Sell	3600	Limit	700	4100	1150063540240	1.60
MSFT	Fidelity	Buy	3600	Limit	700	4800	1150063540207	1.81

Figure 4

everything you need to handle exactly the data that has been modified in the CQ Engine's database and nothing else. A more sophisticated CQ Callback implementation usually builds nested loop logic (rows, then update columns) to handle each delta individually.

The runtime sequence of events from start to finish is as follows:

1. The publishing application receives an event from some external source. In response to the event, it modifies data in the CQ Engine's in-memory JDBC database (i.e., updates a filled quantity on an existing order). The code here is exactly the same as with any non-CQ JDBC database.
2. The CQ Engine gets the update, applies it to the database, and then identifies which Continuous Queries are impacted by the data modification. This is really the magic that makes the entire design pattern successful.
3. The CQ Engine packages row deltas specific to each Continuous Query's requirements and pushes them back to the clients over the JDBC connection. Note that these row deltas are from the client's perspective — meaning that they represent added, modified, and removed items from the client's ResultSet view. An UPDATE statement in the CQ Server database can thus lead to completely different client view deltas, depending on whether the table update caused CQ join conditions to be newly met (causes INSERT RowDelta), dropped out (causes DELETE RowDelta), or if a previously met condition continues to do so (causes UPDATE RowDelta).
4. The CQ Clients update their JDBC ResultSet objects with the deltas and invoke any registered callbacks — in our example, we simply bind the ResultSet to a GUI grid component.

If you download and run this example, you'll see a screen that looks like Figure 4.

Since the Publisher application is continuously updating the CQ Engine database's Quotes table and Orders.FilledQty/Orders.OrderStatus fields, you'll notice that they keep changing several times a second. Thus the example shows how you use a simple JDBC syntax to create a continuously updated view of data and then handle some very straightforward callbacks to execute custom logic in response to data changes. This architecture is capable of scaling to thousands of inbound events/second servicing thousands of Continuous Queries.

Conclusion

Continuous Query technology promises to provide a robust new development model for applications that require complex real-time views of rapidly changing operational data. What makes this approach so exciting is the ability to combine the syntactical power of standard query languages with the performance and scalability required of modern applications.

Although we focused on the basic functional aspects of CQ technology in this article, you'll find that the tools on the market have some very advanced capabilities as well. Features such as CQs with User Defined Aggregations (UDAs), advanced horizontal scale-out, transparent high availability and failover, intelligent flow control, distributed transactions, and much more are either available now or soon will be. This is a technology worth keeping an eye on as you encounter use cases where client views of operational data must be continuously maintained.

Listing 1

```
StringBuilder sql = new StringBuilder ( );
Date timestamp = new Date ( o.timestamp );  // for the
UpdateTimestamp column.

sql.append ( "UPDATE orders SET FilledQty=" );
sql.append ( order.fillQty );
sql.append ( ", OrderStatus = '" );
sql.append ( order.orderStatus );
sql.append ( "', UpdateTimestamp=" );
sql.append ( timestamp );
sql.append ( "WHERE OrderID = '" );
sql.append ( order.orderId );
sql.append ( "'" );

/* For example this might create
UPDATE Orders SET FilledQty=500,
OrderStatus='Partial Fill',
UpdateTimestamp='2006-06-06 12:30:02'
WHERE OrderID = '123456'; */

Statement st = c.createStatement ( ResultSet.TYPE_SCROLL_
INSENSITIVE,
          ResultSet.CONCUR_READ_ONLY );
st.executeQuery ( sql.toString() );
```

Listing 2

```
Properties conxProps = new Properties ( );
// Use whatever database-specific parameters are required here.
conxProps.setProperty ( "endpoints", "server1=localhost:30303" );
c = initJDBCConnection ( conxProps );

// Initialize the Data by creating the necessary tables.  If they
already exist,
// DataPublisherHelper deletes their contents instead.
DataPublisherHelper.createOrdersTable ( c );
DataPublisherHelper.createQuotesTable ( c );

// Initialize orders.  The SimpleOrderGenerator will create a
large basket of
// orders.  We receive a onNewOrder() callback for each one.
orderGenerator.initOrders();

// Start streaming quotes and order fill notifications
orderGenerator.start();
quoteGenerator.start();
```

Listing 3

```
 quick.dbtable.DBTable dBTable1 = null;
java.sql.ResultSet resultSet;

// SQL select to fetch all orders and corresponding quotes for
select financial instruments
static String quotesSql = "SELECT o.Symbol, o.Client, o.Side,
o.Quantity, o.OrderType, o.FilledQty, o.OrderID, o.ClientComment,
q.Bid, " +
     "q.Ask, " +
     "q.AskSize, " +
     "q.BidSize, " +
     "q.UpdateTimestamp " +
     "FROM quotes q, orders o WHERE q.Symbol = o.Symbol " +
    " AND o.Symbol IN ('IBM', 'INTC', 'AMD', 'MOT')";
// NOTE: The "IN clause" above is typically dynamically configured
based on user choice; There will be numerous such views that are
concurrently monitored by the user
// Initialize the GUI table and start the monitor thread
  public void initOrderQuoteTable() throws Exception {
   {
     // set Frame properties
     setSize(1280,1024);
     setVisible(true);

     //create a new quicktable
     dBTable1 = new DBTable();

     //add to frame
     getContentPane().add(dBTable1);

     // to create the navigation bars for the table
     dBTable1.createControlPanel();

     // Start the polling thread to refresh the OrderQuoteTable
     startPollingThread();

    …..

   }

public void startPollingThread() throws Exception {
Thread pollingThread = new Thread( new Runnable() {
public void run() {
 // < Execute the query on DB and obtain instance of java.sql.
ResultSet
 //Refresh the Jtable
 synchronized ( resultSet ) { dBTable1.refresh ( resultSet); }
  pollingThread.sleep (configuredTimeInterval);
 }
  } );
pollingThread.start();
}
```

Listing 4

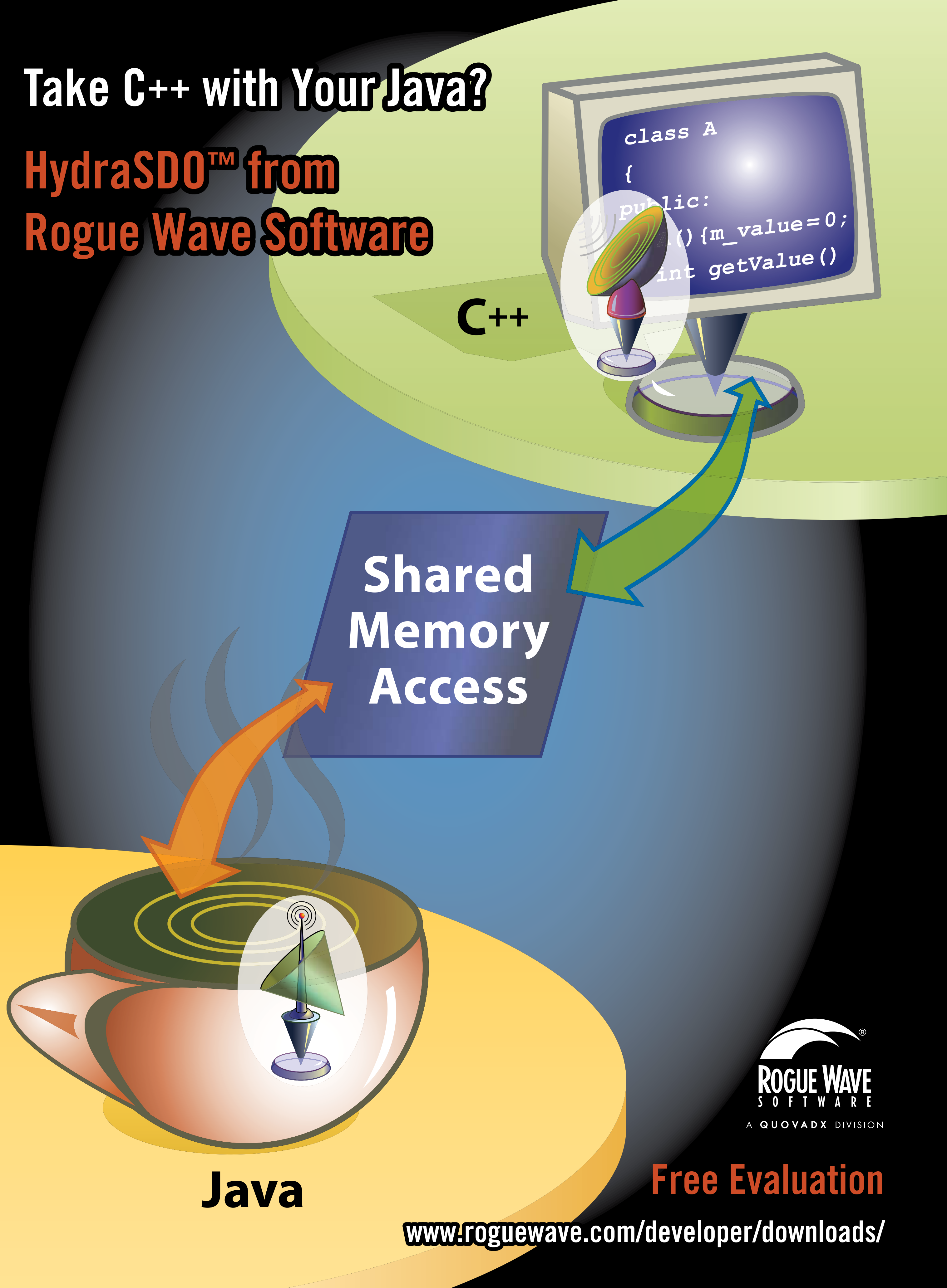
```
// Do the following instead of polling the DB …. Replace startPol-
lingThread above
// Acquire connection of the JMS provider service, start a non-
transactional session on the DB events queue, create a receiver
on the queue and set the listener to receive the events asynchro-
nously.

queueConnectionFactory = jndiLookup(<JMS provider URL, etc>);
queueConnection = queueConnectionFactory.createQueueConnection();

queueSession = queueConnection.createQueueSession(false, Session.
AUTO_ACKNOWLEDGE);
queue = jndiLookup(queueName);

// Use JMS message selector to limit events routed to Jtable cli-
ent process
// Note that 'symbol' would have to be explicitly set as a JMS
header property when message
// is constructed
String selector = "symbol IN ('IBM', 'INTC', 'AMD', 'MOT')";
queueReceiver = queueSession.createReceiver(queue, selector);

dbEventListener = new DBEventListener();
queueReceiver.setMessageListener(dbEventListener);
queueConnection.start();



// Implement the DBEventListener; Implements javax.jms.
MessageListener interface
public class DBEventListener implements MessageListener {

    public void onMessage(Message message) {
      try {
< Execute the query on DB and obtain instance of java.sql.
ResultSet
 //Refresh the Jtable
  synchronized ( resultSet ) { dBTable1.refresh ( resultSet); };
            } catch (JMSException e) {
                <Handle exception>;
            }
        }
    }
```

Listing 5

```
// The SQL we'll use as a Continuous Query
String CQSql = "SELECT * FROM Orders, Quotes WHERE Quotes.
Symbol=Orders.Symbol";

// Create a new QuickTable for visual display of our JDBC
ResultSet
dBTable1 = new DBTable();

// Add the QuickTable to the Swing frame
frame.getContentPane().add(dBTable1);

// Connect to the Real Time Events engine (essentially the same as
any JDBC connection)
Properties conxProps = new Properties();
conxProps.setProperty ( "endpoints", "server1=localhost:30303" );
c = initCQEngineConnection ( conxProps );

// Note that CQManager is specific to the the GemFire RTE CQ
implementation
// Create a CQManager for the Connection
cqManager = CQManager.getCQManager ( c );

// Create an register the continuous query using the CQManager and
our SQL String.
CQ exampleCQ = cqManager.create ( CQSql );
ResultSet rs = exampleCQ.register ( "exampleCQ", CQSql );

// Populate the GUI QuickTable with the Result Set returned by the
CQ initialization.
 dBTable1.refresh ( rs );

// Once the table has been initialized with data, start listening
for CQ Updates  .
exampleCQ.setCQListener ( this );
```

Take C++ with Your Java?

HydraSDO™ from Rogue Wave Software



ROGUE WAVE SOFTWARE®
A QUOVADX DIVISION

Free Evaluation

www.roguewave.com/developer/downloads/

Joe Winchester
Desktop Java Editor

The Death of Mediocrity

Computers can generally be characterized into two types: ones that are designed to have more than one user attached and those intended for a single user. In the beginning almost all computing was done on large multi-user machines, partly due to their expense, which precluded their use to all but large institutions or wealthy corporations. Mainframes ruled this era and excelled at their role: providing a reliable computing platform for hosting databases, transaction servers, and centralized applications. The interaction was through character-based screens that, while providing fast and efficient green screen access, was to be their Achilles heel.

At the other end of the scale are personal computers. PCs have two major benefits over mainframes: a lower cost per unit and the ability to host operating systems with graphical user interfaces. GUI applications make use of event-driven user interfaces that can respond to fine-grained mouse, keyboard, timer and paint requests. This provides the framework on which everything from shoot-em up 3D games, WYSIWYG word processors, business presentations with embedded video, and a plethora of powerful desktop programs reside.

Most of the problems over the 20 years in IT have occurred because one of the two ends of the computing spectrum has tried to venture into the other's domain. PCs tried to become multi-user servers and big iron boxes attempted presentation logic.

The computing section of my local science museum has an exhibit showing black and white photographs from the 1970s with reel-to-reel tape drives, floor-standing disk platters, and wardrobe CPU units filling an operations room. Next to this is a display case with an Altair 8800, the sign teaching us how the smaller machine replaced the room-filling mainframe by matching its computing power at a cheaper cost. The analogy drawn is to that of the dinosaurs, where the large and inefficient behemoths couldn't cope with extreme climate change and died out while smaller and nimbler mammals arose to rule the world in their place.

The rise of PCs is a huge phenomenon where, for most of the 1980s and 1990s, there were more new PCs sold per year than the entire installed base. The prediction by George Moore in 1965 was that the transistor density of semiconductor chips would double every 18 months. This largely held true for the next 40 years, benefiting PCs that continued to double in speed while halving in cost. For those who were in the game of downsizing from mainframes, this enabled them to create server farms by simply daisy-chaining PCs together.

The big iron server guys have always wanted to challenge the rise in PCs, fueled by resentment at the insults of "dumb screen" and "legacy system" that were being thrown at them. The Intern et gave them an opportunity to do this, by enabling them to reinvent themselves as hosts for Web application servers dishing up HTML to clients in place of 3270 or 5250 datastreams.

What has occurred is that PCs have scaled up to become servers and servers have become controllers of presentation through HTML. Both are poor compromises and I think have hurt usability, resilience, and general IT efficiency. The trend in many social systems can be characterized as a pendulum that swings between two extremes, politics being a prime example – once policies become attempted, they fall short of promise and expectations, allowing the previous failed polemic to regain popular traction.

For the fast, nimble PCs that now fill rooms, their fate, ironically, is to be replaced with smaller and faster modern mainframes that outperform them in terms of speed, price, and simplicity. Because of advanced workload management techniques, mainframes can be driven harder, often running at 70–90% utilization, while Wintel boxes typically only manage 5%. While Moore's law held fast for the past 40 years, the runway has run out, as physical laws governing thermal flux prevent any further significant miniaturization. A modern Pentium consumes 100 watts of power and generates more heat per square inch than exists inside a nuclear power station's reactor core. The scalability of a virtualized mainframe is huge, with benchmarks showing that up to 20,000 copies of Linux all running Web servers can co-exist happily inside a single box.

For the PC, what we're seeing now is a growth in applications that exploit its capabilities as a first-class client desktop, rather than a rendering engine for dumb HTML. For Google, the bastion of all things Web, two of their most impressive applications are Google Desktop, which indexes all of a PC's files and provides a set of integrated functions such as chat, to-do lists, phone clients, and for mapping Google Earth offers the ability to walk the earth in 3D making use of the PC's native graphics functionality through DirectX.

What this should spell is a new era in which the two poles of computing go back to basics and rediscover what they're best at is doing what they were designed for. Big multi-user servers will continue to grow in terms of their capacity to become application hosting giants, while PCs will enjoy a period of rich applications that fully exploit their graphics capabilities and provide a high-usability end point. Over the past 20 years the server and the client have fought wars where each has tried to replace the other. What we need for the next 20 is for each to excel at what they're best at, and for users to benefit from faster, easier, and richer software.

Joe Winchester is a software developer working on WebSphere development tools for IBM in Hursley, UK. *joewinchester@sys-con.com*

IT'S THE END OF THE LINE FOR SPAM.

PUT IT IN THE CAN.

CanIt
CONTENTS: JUNK E-MAIL

**Ads, fraud schemes, porn, viruses, phishing attempts…
if it's offending e-mail, Roaring Penguin Software's CanIt™
solutions will stop it before it reaches your mail server.**

- Simple administration
- Automatic spam and virus updates
- Compatible with all e-mail servers
- Available as software or hardware appliance
- Flexible end-user controls

**Throw the book at spam today!
Request a free 20-day CanIt evaluation
and stop spam on YOUR terms!**

ROARING PENGUIN
SOFTWARE INC.

www.roaringpenguin.com +1 (613) 231-6599 sales@roaringpenguin.com

Managing a Standardized Build Process Outside of the Eclipse IDE

Point-and-click solutions won't cut it

by Steve Taylor

Building objects in the Eclipse IDE is simple – it's a point-and-click solution. However, as applications built on the Eclipse platform mature the need for building outside of the IDE increases. This need can be driven by the development team that is striving to perform agile development techniques where builds are executed based on a file "check-in" action into an SCM tool. The need can also be driven by IT governance where a scheduled and audited production build is required. Moving from builds managed inside of the Eclipse platform to builds managed outside of the Eclipse platform can be a big task in itself. Don't hesitate to make this jump. It's a jump that you'll find you can't do without. The sooner you get out of your point-and-click build process, the sooner your application will begin to mature.

Defining the build process should not be taken lightly. Auditability and traceability of built objects are becoming increasingly important due to IT compliance mandates. This means that your builds must become more traceable than a point-and-click process. Don't make the mistake of addressing the issue of building outside of the Eclipse IDE long after the application has grown to an unmanageable size. Delaying the inevitable only results in a poorly managed, unplanned, ad hoc build process that isn't sustainable, can't meet IT compliance, and involves expensive hidden costs in maintenance and fixes.

There are three ways of addressing the build process outside the Eclipse IDE. The most common method is to manually develop and maintain Ant/XML scripts. These scripts use Ant Tasks from the Apache Foundation to act as a wrapper to the Java compiler. The second method is to write scripts that call Eclipse in what's called a "headless" mode. A script that executes the build in headless mode acts in the same way as the point-and-click process inside the Eclipse IDE, but does the build from a script. And finally, the preferred method is to use a commercial build tool that can automate the creation of the scripts. Commercial tools that minimize the use of manual scripts establish a more repeatable and traceable build workflow, the ultimate goal of any solid development process.

If you don't have the luxury of a commercial build tool that can create a solid reusable build framework, you must create a manual build process that's as standardized as possible. A "manual" build process refers to any scripted build process that has to be maintained manually. Even if you execute your manual scripts through a job scheduling build management tool, your builds are still manual because you must manually maintain the build logic contained in the scripts.

When writing the manual build process, your choices become writing Ant/XML scripts to perform the build or to use the Eclipse headless mode option. Establishing a repeatable build that can be traced requires that you concisely define how the build executes and with what source code. The use of the headless mode removes that level of control. It provides a little more functionality than using the point-and-click process inside Eclipse. When running a headless mode script, you're still relying on the Eclipse IDE to control the build. For this reason, defining a build process using Ant/XML is recommended if a commercial tool isn't available.

If you review any Ant/XML script, it may appear that the process of converting Java source into Java jars is complicated. This isn't necessarily true. Ant/XML files execute in a serial fashion, top to bottom. Everything must be precisely coded in a particular order. That's why XML build scripts can be very large and difficult to debug. There are some suggested standards on writing XML scripts, but they're not always followed. At a minimum, XML build scripts should follow a basic flow with pre-processing and post-processing steps that are consistent for every XML script created, planned, or unplanned.

Pre-Processing Steps

Pre-processing steps are used to establish the environment in which the subsequent task will execute. The point of setting up these pre-processing steps is to get the source code and variables organized and do overall housekeeping before compilation starts. A common mistake is to mix together these housekeeping steps for each call to the Java compiler or before calling an Ant Task. By organizing the pre-processing steps at the beginning of the XML scripts, the build process becomes clearer and easier to follow. It also reduces redundancy and results in an improved, more efficient process. These are the recommended pre-processing steps:

ClassPath

The CLASSPATH identifies which Java Classes are going to be used to resolve inter-class dependencies. The Java compiler will search the CLASSPATH in a first-found method and use a file as soon as it's been located. The CLASSPATH can include Jar and Zip files and directories.

Steve Taylor is president and CTO of Catalyst Systems Corporation. He is a senior developer with 19 years of experience in both distributed and mainframe application development. Prior to founding Catalyst in 1995, Steve served as a technical consultant assisting companies with defining a solid build and release process. In this capacity, he became expert in the use of configuration management and release tools and recognized the need for a solid, reusable, and repeatable build process. At this time he began developing the build procedures that have since become Openmake. Steve got his BS in computer science/mathematics from the University of Illinois-CU.

When setting the CLASSPATH it's important to make sure of two things. First, the CLASSPATH should only be defined once in your process. Second, only the jar files and class directories that are used should be referenced in the CLASSPATH. Don't reference other unused jars since then the Java compiler will do more work than needed, slowing down your build substantially. And specifying only the used jars will make for quick dependency identification. There's nothing worse than attempting to trace jar file dependencies only to find that a large number of jar files aren't needed. This can add substantial time to debugging your builds.

The use of wild cards is always a topic of debate. Wild cards can eliminate typing in your script, but in the end may cause your build process to include more objects than needed. List each jar file in the CLASSPATH explicitly to prevent any vagueness. This is particularly critical when exposing your build details for IT compliance mandates. Wild cards aren't traceable.

Setting the CLASSPATH should be the first task in the Ant XML script.

Copying and Renaming Files

There are cases in which source code, jar and property files have to be copied from one location to another for the compiler to find the file or put it in the archive correctly. Do yourself a favor and minimize the use of copying and renaming of files. It makes it extremely difficult to trace the archive contents back to the original source. Copying files around also creates a more "magical" build process. IT compliance mandates want a clear view into your process. No "magic" is needed.

Instead of copying or renaming the files, put the files in the correct location from the start. Don't use your Ant/XML script to clean up a mistake in file organization. This may involve updating your project directory structure and making your Source Code Management tool more efficient. As an alternative to copying and renaming files, the use of the Ant Task "Zip" and its attributes, such as "dir" and "prefix" can handle getting source from one location and putting it in the archive at a different location.

This sample XML code from the Apache Ant Manual demonstrates using the Ant Task "Zip" to take one source location (htdocs/manual) and put it in another location (docs/user-guide):

```
  <zip destfile="${dist}/manual.zip">
    <zipfileset dir="htdocs/manual"
prefix="docs/user-guide"/>
    <zipgroupfileset dir="."
includes="examples*.zip"/>
  </zip>
```

Compile Step

The compile step is, of course, the heart of your process. It will become the largest section of XML script. The important point to remember when defining this portion of your script is the management of dependencies.

Dependency References

With Ant, you can explicitly define the dependencies between tasks. For example, the JAR task can be dependent on the JAVAC task. Ant will also let multiple task dependencies be established. Don't be seduced by this seemingly convenient Ant Task. While it seems useful, it can be burdensome. When tracing the order of executing the various Ant tasks in the Ant/XML script, it's much easier to follow a dependency chain that has only one task dependency instead of multiple ones. For example:

Scenario 1
JAR Task depends on JAVAC Task
JAVAC Task depends on the COPY Task
COPY Task depends on the INITIALIZATION Task
Versus

Scenario 2
JAR Task depends on the JAVAC Task and COPY Task
JAVAC Task depends on the COPY Task and INITIALIZATION Task
COPY Task depends on the INITIALIZATION Task

As you can see in Scenario 2, there are redundant Ant task dependencies. For example, the COPY Task is redundant on the JAR Task. This redundant use of COPY Tasks isn't needed since it's already referenced higher up in the task dependency hierarchy being the JAVAC Task.

There will be cases when you want to have multiple task dependencies as in the creation of a war file. In this case, multiple task dependencies may be needed to ensure that all of the jars are created before the war. But each jar should have just one task dependency, that being the JAVAC task.

Identifying Source Code

Finding your source can seem like an easy item at first, but when applications get bigger there's a greater chance that wrong or obsolete code is included. Using wildcards in the Ant/XML script is an easy way to minimize the need for typing, but for the wildcards to be effective, the source files have to be efficiently organized in a proper Java package directory structure.

The best way to manage source is to define an efficient package directory structure. So you must move beyond your unique needs and address the package names at a more global level in your organization. It's best to make sure that a corporate Java package structure is agreed on and used. As part of the Java package structure it's best to keep the package names simple. Really long package names can cause problems with the file limits on the Windows operating system. Java compiles on Windows have been known to stop working when the 254-character limit is exceeded. To make this problem even peskier, the script may work on one user's machine but break on another's. This is due do the build directory root name being added to the package names. For instance, one person may do the build in c:\mybuilds but another may build in d:\onlinedata\j2ee\development\code. The difference in the directory name can make or break the build by pushing the 254-character limit.

Another item that defines the location of source code is the use of the excludes attribute of the JAVAC Ant Task. It's best to remove the older obsolete code from the Source Code Management tool and from the file system instead of using the exclude attribute. Most SCM tools provide for renaming or removing an item without loosing all of the history. SCM tools also allow for comments that create a level of traceability on why a piece of code is no longer required. Having this information in the SCM tool makes for easier access versus the information being hidden in a comment in the Ant XML.

JAVAC –sourcepath

The native command line Java compiler (javac.exe) has an interesting flag called –sourcepath that provides a directory concatenation to find the source code. It works on a first-found basis. So once the source has been located, the directory browse stops. There are two advantages to using this parameter. First, all of the code doesn't have to be found in the current build directory. That is, source code can be found in multiple locations. Thus, the build process only has to check out the changed code and find the remaining code from a previous full checkout. This will speed the overall build process by minimizing the files to be checked out. Second, if the JAVAC command is given a Java file as a parameter it will then check using the –sourcepath parameter for additional source that's referenced by the original Java file and compile it too. This process will allow just the changed source to be passed as the parameters to JAVAC and JAVAC will figure out all the remaining dependencies for you.

Post-Processing Steps

As with the pre-processing tasks, the post-processing tasks should avoid items such as copying and renaming files. If an archive has to be given a specific name then that name should be handled on the archive task instead of doing a copy or rename. The use of multiple task dependencies should also be minimized to ensure easy traceability and IT compliance.

Testing

One of the common items to do in the post-processing phase is to test. These tests are usually unit tests such as Junit. But these tests can also include some basic tests about the archive itself, such as checking to see if the correct deployment descriptor and properties files have been used or checking the number of files in the archive to verify at a basic level if all of the source was compiled. Another useful test is to check to see if the archive contains the correct manifest and directory structure. Validating these items before deployment can save you the embarrassment of a production failure. It's best if these items are extracted and e-mailed to a tester to verify their accuracy before deployment.

Deployment

The deployment task should be one of the last steps in the build process. It should be dependent on the Testing task and the Testing task dependent on the Build task. But it shouldn't be the default task that gets executed. You want to give the user the option of just building and testing or building, testing and deploying. This lets someone just deploy.

Variable

Beyond the pre-processing, compiling, and post-processing steps in your Ant/XML scripts, managing variables is also critical in creating a more standardized manual process.

Using variables lets an Ant/XML script be written to execute on multiple machines. This is a cautionary tale, however, because using too many variables makes an Ant/XML script hard to read and debug. It's best to use variables for the directory path on the Jar files in the CLASSPATH and for the source code locations. For example, instead of using:

```
 c:\jdk2\lib\rt.jar you would use
${JAVAHOME}\lib\rt.jar
```

This reference will let different users have different working locations. Again, the directory structure of the source code and libraries should be laid out efficiently for builds.

Machine-Specific Variables

If there are any machine-specific items referenced in the Ant/XML script then they should be referenced through a variable and abstracted out. When you write the Ant/XML script assume that you won't be the only one using the script.

By following a standard guideline, your Ant/XML scripts can become easier for another developer in your organization to follow and so more traceable. This is ultimately what you're striving for. Traceability in your build process can only be achieved if someone else can follow the build steps. By maintaining some standard sections such as pre-processing, compiling, and post-processing, your scripts should follow a basic structure that can be easily identified and traced.

Commercial Eclipse plug-ins are available that can substantially minimize the need for Ant/XML scripting. These tools provide a reusable build framework through a standardized interface.

Commercial build tools that simply execute your Ant/XML scripts may be helpful in managing the many scripts that are created over time; however, tools that minimize your scripting effort are preferable because they create a solid reusable framework once that can be reused over and over.

The inherent problem of Ant/XML scripting is that the scripts are written for one jar and one application at a time. This creates a lot of redundancy. Redundancy equates to higher cost and lower quality. Just as you strive for reuse when developing applications, you should strive for reuse in your application build framework. Using scripts to do this is close to impossible because manual Ant/XML scripts contain hard-coded application references.

Commercial tools such as Openmake by Catalyst Systems Corporation, Perfect Build by CodeFast, and Builder by Serena address the scripting issue directly by providing a reusable framework in your build process.
Open Source tools such as Maven will also assist you in minimizing the amount of scripting necessary for each jar file you create.

As upper management demands more accountability from the development process, the build component will be scrutinized more closely. A point-and-click process from the Eclipse IDE won't meet the new IT mandates. Neither will overly complicated nor non-standardized build scripts. Eventually you'll be forced out of the comfort of your point-and-click IDE and into a more standardized method. Your choices will be to rely on Open Source languages such as Ant/XML and a lot of hard work or a commercial tool to help you with the job. Regardless of your future build requirements, the effort in creating standards for the build is critical and well worth the effort.

References

- www.apache.org – Learn about Ant Tasks, Ant scripting and Maven
- www.openmake.com – Learn about reusable scripts with Openmake
- www.codefast.com - Learn about script generation with Codefast

REALWORLD FLEX SEMINAR

www.flexseminar.com

Learn How to Build the Next Generation of Web Apps **from the Experts!**

August 14, 2006
The Roosevelt Hotel
New York City

Adobe Flex 2 is a complete, powerful application development solution for creating and delivering cross-platform rich Internet applications (RIAs), within the enterprise and across the web. Not until now has there been a way for enterprise programmers and architects to work with existing tools of choice, familiar programming models and integration with existing systems and infrastructure. Multi-step processes, client-side processing, direct manipulation and data visualization are all key factors in the Flex solution.

The "Real-World Flex" One-Day Seminar will delve deep into the central workings of Flex so that the seminar delegates can integrate this timely new technology into their applications, creating powerful interactive content. Delegates will learn from the experts who not only created the product but also those who are using it to the massive benefit of their employers, their customers and the Web.

"Go Beyond AJAX with Flex."

The list of topics at the Real-World Flex Seminar includes:

- ✓ ***The Flex Approach to RIA Development***
- ✓ ***Bridging Flex and AJAX***
- ✓ ***Integrating The SPRY Framework***
- ✓ ***Using Flex Builder 2***
- ✓ ***Flex for Java Developers***
- ✓ ***ActionScript 3.0 Tips and Tricks***
- ✓ ***How To Use ColdFusion with Flex***
- ✓ ***Leveraging Flash***
- ✓ ***Preparing for Adobe Apollo***
- ✓ ***MXML Master Class***

Who Should Attend?

- ✓ ***CEOs and CTOs***
- ✓ ***Senior Architects***
- ✓ ***Project Managers***
- ✓ ***Web programmers***
- ✓ ***Web designers***
- ✓ ***Technology Evangelists***
- ✓ ***User Interface Architects***
- ✓ ***Consultants***
- ✓ ***Anyone looking to stay in front of the latest Web technology!***

Early Bird:
(Hurry for Early Bird Pricing) $395*

Conference Price:
(Register Onsite) $495*

OFFER SUBJECT TO CHANGE WITHOUT NOTICE, PLEASE SEE WEBSITE FOR UP-TO-DATE PRICING

* Golden Pass access includes Breakfast, Lunch and Coffee Breaks, Conference T-Shirt, Collectible Lap-Top Bag and Complete On-Demand Archives of sessions in 7 DVDs!

Sponsored by Adobe Web Developer's & Designer's Journal

Produced by SYS-CON EVENTS *The World's Leading i-Technology Event Producer* www.events.sys-con.com

Rich Internet Applications: AJAX,

www.AjaxWorldExpo.com

AjaxWorld™ CONFERENCE & EXPO

SANTA CLARA SILICON VALLEY

SYS-CON Events is proud to announce the first-ever AjaxWorld Conference & Expo 2006!

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

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

CALL FOR PAPERS NOW OPEN!

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

Flash, Web 2.0 and Beyond...

REGISTER TODAY AND SAVE!

October 3-4, 2006

Santa Clara Convention Center
Hyatt Regency Silicon Valley
Santa Clara, CA

To Register
Call 201-802-3020 or
Visit www.AjaxWorldExpo.com

May 7-8, 2007
First International AjaxWorld Europe
Amsterdam, Netherlands

"Over the two information-packed days, delegates will receive four days' worth of education!"

Early Bird*
(Register Before August 31, 2006)
.... $1,495**
See website or call for group discounts

Special Discounts*
(Register a Second Person)
.... $1,395**
See website or call for group discounts

(5 Delegates from same Company)
.... $1,295/ea.**
See website or call for group discounts

On-Demand Online Access
(Any Event)
.... $695

*Golden Pass access includes Breakfast, Lunch and Coffee Breaks, Conference T-Shirt, Collectible Lap-Top Bag and Complete On-Demand Archives of sessions in 7 DVDs!

**OFFER SUBJECT TO CHANGE WITHOUT NOTICE, PLEASE SEE WEBSITE FOR UP-TO-DATE PRICING

"It Was The Best AJAX Education Opportunity Anywhere in the World!" —John Hamilton

Topics Include...

Themes:
> Improving Web-based Customer
> Interaction
> AJAX for the Enterprise
> RIA Best Practices
> Web 2.0 – Why Does It Matter?
> Emerging Standards
> Open Source RIA Libraries
> Leveraging Streaming Video

Technologies:
> AJAX
> The Flash Platform
> The Flex 2 Framework & Flex Builder 2
> Microsoft's approaches: ASP.NET, Atlas, XAML with Avalon
> JackBe, openLaszlo
> JavaServer Faces and AJAX
> Nexaweb
> TIBCO General Interface

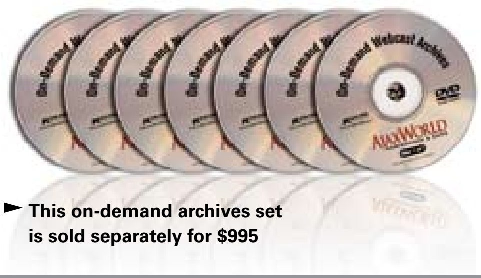
Verticals:
> Education
> Transport
> Retail
> Entertainment
> Financial Sector
> Homeland Security

GROUP DISCOUNTS AVAILABLE:
— 5 Delegates from Same Company — for only $995 (each)
— Register a Second Person — for only $1195

LIVE SIMULCAST!
AROUND THE WORLD ON SYS-CON.TV

Receive FREE WebCast Archives of Entire Conference!

The best news for this year's conference delegates is that your "Golden Pass" registration now gives you full access to all conference sessions. We will mail you the complete content from all the conference sessions in seven convenient DVDs after the live event takes place.



► **This on-demand archives set is sold separately for $995**

SYS-CON EVENTS
For more great events visit www.EVENTS.SYS-CON.com
VISIT WWW.AJAXWORLDEXPO.COM FOR THE MOST COMPLETE UP-TO-DATE INFORMATION

JDJ Editors' Choice Awards

The editors of Java Developer's Journal are in a unique position when it comes to Java development. All are active coders in their "day jobs," and they have the good fortune in getting a heads up on many of the latest and greatest software releases. They were asked to nominate three products from the last 12 months that they felt had not only made a major impact on their own development, but also on the Java community as a whole.

The following is a list of each editor's selections and the reason why they chose that product.

Joe Winchester
Desktop Java Editor

SwingLabs

SwingLabs is an open source laboratory for exploring new ways to make Swing applications easier to write, with improved performance and greater visual appeal. It is an umbrella project for various open source initiatives sponsored by Sun Microsystems and is part of the java.net community. Successful code and concepts may be migrated to future versions of the Java platform.
http://swinglabs.org

Everything that has come out of SwingLabs – this is an absolutely fabulous open source project that allows skunk work–type development to occur outside of the JCP that then gets rolled back into the Java Standard Edition. It has created superb frameworks like the Timing framework to allow crisp and elegant animation effects, the SwingX project that has spawned fantastic new widgets, and APIs including JXPanel and the whole concept of painters, as well as nice high-level work like the data binding project to allow easy GUI to data connectivity.

The Eclipse Rich Client Project

While the Eclipse platform is designed to serve as an open tools platform, it is architected so that its components could be used to build just about any client application. The minimal set of plug-ins needed to build a rich client application is collectively known as the Rich Client Platform.
http://wiki.eclipse.org/index.php/Rich_Client_Platform

This is just an awesome technology that allows Java developers to leverage the core plumbings of Eclipse, namely OSGi, SWT, JFace, and other frameworks, to create their own desktop application. It's already being used very successfully by a large number of clients and goes from strength to strength, making it a powerful way for people to build extensible desktop applications. I think it has the potential to really change the way Java client applications are built.

The Java Web Start Improvements for Mustang

Using Java Web Start technology, standalone Java software applications can be deployed with a single click over the network. Java Web Start ensures the most current version of the application will be deployed, as well as the correct version of the Java Runtime Environment (JRE).
http://java.sun.com/products/javawebstart/

One of the big, possibly only, reasons why users today must suffer the poor usability of "dumb" browsers is because distributing and maintaining proper client apps is difficult. HTML makes this ridiculously easy and is a good engineering solution, but one that offers very poor end usability. JWS was always the promised savior to allow desktop distribution over HTTP but never really lived up to its expectations in previous releases. With the Mustang work now it looks very, very good, though with many of the dialogs simplified; better looking; and it seems like it's finally going to allow first class, easy and polished large-scale distribution of Java clients to help rejuvenate Java on the desktop.

Yakov Fain
Contributing Editor

Adobe Flex 2

Adobe Flex 2 is an application development solution for creating and delivering cross-platform Rich Internet Applications (RIAs) within the enterprise and across the Web. It enables the creation of expressive and interactive web applications that can reach virtually anyone on any platform.
http://www.adobe.com/products/flex/

Adobe Flex 2 is a very potent player in the Rich Internet Application arena. Flex 2 is a direct competitor of Java Swing and AJAX. It offers declarative programming and a rich library of cool-looking and functional components. Your compiled code runs in a Flash 9 virtual machine. Flex 2 offers fast protocols for data exchange with the server-side components, server push, data binding, easy integration with Java, JMS support, and more. I was very impressed.

IntelliJ IDEA

IntelliJ IDEA is a Java IDE focused on developer productivity. It provides a combination of enhanced development tools, including refactoring, J2EE support, Ant, JUnit, and version controls integration.
http://www.jetbrains.com/idea/

This Java IDE is the best available today. Despite the fact that it's not free (the price is very modest though), IntelliJ IDEA has a loyal following of Java experts who can appreciate the productivity gain this tool brings for a small price. Finding classes, refactoring, suggesting solutions, even a JavaScript editor for AJAX warriors…everything is at your fingertips. The upcoming version, 6.0, will include a new UI Designer and Google Web Toolkit support.

WebCharts 3D

WebCharts3D is a development toolkit that offers flexibility for all aspects of rich-client and Web-based charting requirements and provides a single-source solution for data visualization.
http://www.gpoint.com

This is one of the best charting components available for Java applications. It's easy to learn and integrate with your Swing, JSP, and JSF applications. The product provides a rich set of charts, gauges, and maps, and can generate not only binary streams but also HTML, which makes it a good choice for AJAX applications. For Web applications, deployment consists of adding one JSP and copying one library to WEB-INF/lib.

Jason Bell
Contributing Editor

Head First Design Patterns by Elisabeth Freeman, Eric Freeman, Bert Bates, and Kathy Sierra (O'Reilly Media)

Using the latest research in neurobiology, cognitive science, and learning theory, Head First Design Patterns will load patterns into your brain in a way that sticks; in a way that lets you put them to work immediately; in a way that makes you better at solving software design problems, and better at speaking the language of patterns with others on your team.
www.oreilly.com

Without doubt the most effective book I have ever read and extremely easy to read. Don't be fooled by the comical light-hearted way this book looks. The chapter with the intro RMI is the best I've ever come across. All the other design pattern books fade into the distance in my opinion.

NetBeans 5

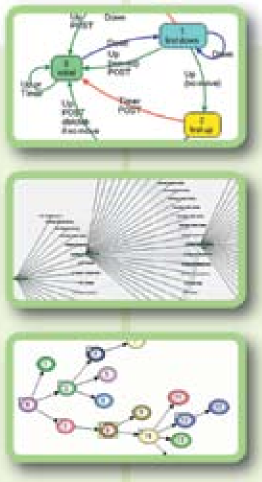
NetBeans IDE 5.0 includes comprehensive support for developing IDE plug-in modules and rich client applications based on the NetBeans platform. NetBeans IDE 5.0 is an open source Java IDE that has everything software developers need to develop cross-platform desktop, Web, and mobile applications straight out of the box.
www.netbeans.org

After a bit of a love/hate start with NetBeans I've now become a convert. It's very easy to use and the enterprise support is excellent. It would be nice to see coverage of the "other" app servers such as Orion and Resin but that's a small price to pay. An excellent product.

A4 Journal and a Ballpoint Pen

For me everything starts on paper, whether it be sketch drawings and UML diagrams. I've never mentioned it over the years but I'd be really lost without it. I've had the delight of looking back through my journals of the past five years and seeing how I've developed and how my ideas have developed with it.

Build interactive diagrams easier than you ever imagined



Create custom interactive diagrams, network editors, workflows, flowcharts and design tools. For web servers or local applications. It's easy-to-use and very flexible. We're the first developer of diagramming components and still the best. Find out for yourself: download our FREE fully functional evaluation kit, with full support at www.nwoods.com.

FREE Download With Full Support!

GoDiagram

Interactive diagram components

www.nwoods.com

The JCP Program: Beyond the 300 Mark

by Onno Kluyt

JavaOne has a catalyzing effect on Java developers: their enthusiasm and energies spike around the show; they ready their latest and greatest Java technology–based projects and solutions for the annual encounter with software programmers from around the world. Take for instance the JSR Spec Leads – they too intensify their efforts around the show to submit new JSRs to the program, advance work under development to the next stages, or finalize standards. JavaOne is a favorite event with JSR Spec Leads who don't miss on the opportunity to leverage the Conference as an ideal forum for sharing their accomplishments and forays into new Java standards projects with their fellow developers. The show's 2006 edition was no exception. Here are the JSRs that brought the JCP Program closer and closer to the 300 mark and crossed it in less than a month.

Modularity in Java is tackled by JSR 294, Improved Modularity Support in the Java Programming Language, led by Gilad Bracha, Sun Microsystems. The project sets out to extend the Java programming language with new constructs that allow hierarchical modular organization. The Spec Lead and Expert Group expect these constructs to be supported at the virtual machine level, through modifications or extensions to the JVM's access control rules. If you are interested in modularity in Java, check out the JSR page, contact the Spec Lead for more information, or send your comments to jsr-294-comments@jcp.org

Another JSR submitted around the same time is JSR 295, Bean Binding. It aims to define an API that greatly simplifies connecting a pair of JavaBean properties to keep them in sync. As proposed by the Spec Lead, Scott Violet of Sun, the connection is intended to be configurable with type conversion and validation operations being able to be applied before updating a property. Bean Binding will be developed so that it reduces the amount of tedious and error-prone code JavaBean developers must write by making additions to the JavaBeans API that up-level connecting pairs of JavaBean properties.

The May marathon of new JSR proposals continued with JSR 296, Swing Application Framework introduced by Sun with Spec Lead Hans Muller at the helm. This JSR commits to providing a simple application framework for Swing applications. It will define infrastructure common to most desktop applications. In so doing, Swing applications will be easier to create. The experts working on it anticipate supporting implementations for current Java releases as well as Java SE 7 (code name "Dolphin").

The next JSR submitted at the time of JavaOne is JSR 297, Mobile 3D Graphics API 2.0, and is targeted at Java ME. It was introduced by Nokia and is shepherded by Spec Lead Tomi Aarnio. The specification is a new revision of M3G (JSR-184), which plans to expose the latest graphics hardware features on high-end devices while improving performance and memory usage on the low end. The submission of this JSR was prompted by the needs of developers, device vendors, operators, and consumers who are looking for richer, smoother, more realistic graphics for games and user interfaces, as the JSR page highlights that I recommend you visit if you're interested in keeping abreast of the latest in the area of Mobile 3D Graphics.

A relatively new member of the JCP, SK Telecom Co. has already embarked on developing a JSR for telematics, JSR 298, Telematics API for Java ME. The proposal, currently under reconsideration ballot, sets out to define the API set for Telematics Service on mobile devices. A Java-based telematics standard could facilitate the introduction of new value-add services related to car management ranging from 911 emergency calling to complex driving guidance. SK Telecom believes that by providing a uniform API set designed for embedded devices, the existing telematics solutions and services can be modified in a more interoperable way and a variety of new Java-based telematics services will emerge more easily. If you are into telematics or aspire to enter this field, check out the JSR page and send your comments to jsr-298-comments@jcp.org

An active participant of the JCP, JBoss came forward around the same time with a project that became JSR 299, Web Beans API. The Spec Lead is Gavin King, a seasoned lead and expert group participant, who will drive the development of the specification to accomplish a standard that unifies the JSF managed-bean component model with the EJB component model. The result of this work is hoped to be a significantly simplified programming model for Web-based applications. The specification promises at the end of the standardization work to provide a programming model suitable for rapid development of simple data-driven applications without sacrificing the full power of the Java EE 5 platform. If you've been following the evolution of the EJB and persistence standards, make sure you check out this proposal too.

The JSR sprint continued after JavaOne. LG Electronics submitted DRM API for Java ME shortly after the conference and with it the first JSR in "the 300 series." Dnyanesh R. Pathak, the JSR Spec Lead and the supporting Expert Group, will work to define an optional package for developing Java ME applications that utilize or interoperate with DRM agents that separately exist in devices. The proposed JSR commits to providing standardized support for digital content protection and management of the rights by enabling APIs to interact with the underlying DRM agent(s). Developers will be able to use this JSR for interacting with the DRM agents for developing applications that handle DRM-protected content.

JSR 301, Portlet Bridge Specification for JavaServer Faces from Oracle followed in July, showing that there's no let up when it comes to Java developers' enthusiasm and dedication to Java technology even if it comes head-to-head with heat waves or tempting vacation plans. Michael Freedman will lead this project, which attempts to standardize the behavior of bridge implementations to ensure true interoperability for JavaServer Faces artifacts. At the time of writing the JCP Executive Committee (EC) still have to vote this submission as a JSR, but once it's approved, if you want to participate in its development I encourage you to contact the spec lead.

The JSR EC ballot of July 24 will carry two more interesting proposals. One is JSR 302, Safety Critical Java Technology from The Open Group to be led by Douglass Locke. The project proposes to create a Java ME capability, based on the Real-Time Specification for Java (JSR-1). The proponents argue that safety-critical systems need a certifiable (e.g., DO-178B) Java environment. Certifiability implies hard real-time resource management and generally very small implementations with low complexity. The existing Java ME and RTSJ (JSR-1) specifications contain too many and too complex functions to render them certifiable. For example, Java ME and the RTSJ assume the presence of a garbage collector; the proposed specification will not assume the presence of a garbage collector.

The other project on the ballot is JSR 303, Bean Validation. The specification intends to define a metadata model and API for JavaBean validation and will not be specific to any one tier or programming model. The Spec Lead, Jason Carreira, views this API as a general extension to the JavaBeans object model and as such expects it to be used as a core component in other specifications, such as JavaServer Faces, Java Persistence API, and Bean Binding. This standardized validation metadata and standard validation API will be valuable across a number of application domains, from Swing desktop applications to Web applications and the persistence layer. The intention is also to deliver this JSR as a component of Java SE 7 (code name "Dolphin") and develop an implementation of the spec as a public open source project, either at java.net community or the Apache Software Foundation.

As I'm signing off, I'm doing a final check of JSR submissions and, yes, a new proposal has just come in, this time a Java ME project from Motorola and Ben Q Corporation, JSR 304, Mobile Telephony API version 2. The proponents plan to take this specification beyond JSR 253 (Mobile Telephony API) and include support for some technologies such as VoIP. They also intend to address some aspects related to control of video telephony sessions.

Forget the summer blues, Java developers keep cool projects coming – 11 new proposals in just a few weeks – which merit keeping an eye on. Stay tuned for more in "the 300 series."

Onno Kluyt is director of the JCP Program at Sun Microsystems and Chair of the JCP. onno@jcp.org

Unofficial History

– continued from page 3

AOP stands for aspect-oriented programming. It's an interesting concept that allows you to change the behavior of a compiled application without changing its source code. For example, you can implement a cross-cutting concern like logging after the application was written and turn it on or off as needed. AOP definitely will be used in some applications, but it's not going to revolutionize programming as OOP did 15 years ago.

The latest fashionable thing is AJAX – a self-proclaimed savior of Web applications. You enter a letter in an HTML search text field , and the results comes back without the page refresh. Time will show if AJAX is the right solution for Web 2.0, but many vendors are trying to make their tools AJAX-enabled because it sells well today.

Meanwhile Java developers go crazy, because of this orgy of 50+ Web frameworks that do the same thing as Struts.

During the last three to four years, lots of enterprise mission-critical systems were moved from the Unix to the Linux platform, and this trend will continue.

Ruby on Rails is heavily promoted by a group of enthusiasts. At this point it's not clear if Ruby will become a commercial programming language, or just another good language such as Lisp or Smalltalk. I don't know, but I'm planning to purchase a book about this language.

Rich Internet Application are back; I'm talking about fat clients here. The major players are Adobe Flex 2, Microsoft WPF, and Java Swing with JWS, of course. This is an interesting field to be in today.

What about us programmers? We have to keep learning more and more buzzwords/tools/frameworks/languages to become senior software developers…oops, I meant to say architects. Why not developers? Because only architects can possibly figure out how to put all these unrelated pieces together.

I want back in the '90s…seriously.

Open Source Design Tools

– continued from page 6

tool that struck the dual sweet spot of automating menial work while enabling me to remain creative. Ever since then, I've worked on EMF in my day job and never looked back.

Given my own evolving ideas, these days, even when I hear "well considered" objections that sound all too similar to my own, I am quite certain that the value of modeling will slowly but surely become clear. It's ultimately not the modeling tools that are of the greatest value, but rather the models themselves.

Ed Merks is co-lead of the top-level Eclipse Modeling Project as well as the lead of the Eclipse Modeling Framework Project. He has many years of in-depth experience in the design and implemention of languages, frameworks, and application development environments. He has a PhD in computing science and is a co-author of the authoritative Eclipse Modeling Framework, A Developer's Guide (Addison-Wesley, 2003). He works for IBM Rational at the Toronto Lab.

Letters to the Editor

My Observations

["RIA with Adobe Flex 2 and Java"
by Yakov Fain, Victor Rasputnis,
and Anatole Tartakovsky Vol. 11, issue 5]

I have been working both with Java Swing and ActionScript while creating a GUI. Here are my observations that defy your statements in the article.

1. "Imagine the amount of Java code you'd need to write to achieve the same functionality." There are a number of Java XUL implementations and Swixml is one of my favorites.
2. "...but we wouldn't have to worry about routing all events to the event-dispatch queue." In Java, you use listeners and handler functions to attach to the GUI events. Events are routed automatically. Creating custom events in ActionScript would take just as much effort as in Java (or probably less, since it has Observer,Observable and other utility classes in the rt library, unlike Flash).

Now the drawbacks of using Flash (components v.2) over Java:

1. Flash components are badly written; there are many undocumented bugs that you would never overcome, e.g., try adding a combobox to an accordion pane, or a menu inside of a scrollpane. The most awful truth about Flash components is that they are badly integrated with one another and putting them inside one another will most likely result in something quite unpredictable (only frozen layers from a component can be seen; the focus frame cannot be set; dropdown layers are displayed underneath another nearby components, etc.)
2. Flash components are closed source. Even if you care to dig into the truth, you wouldn't risk changing anything because the bug is in the layered structure of the Flash drawing and there are some constraints on using the depths of those layers.
3. There are no skins for different components, and it's unlikely that Adobe will come up with something unbuggy in the next release, since their main interest is increasing the feature set and popularizing some visual benefits, but it's hell for programmers. (Some 4+ releases in my history have proven that at least to me.).
4. Flash has no threads, no thread management. If you start some calculation or even a simple data manipulation or object creation during some visualization process, you get a freezer.

All the rest about the small size, video/audio, Web integration, cross-platformedness is true, but I wouldn't use Flash in a project with complex GUI.

—Vitaly Sazanovich

Vitaly,

Thank you for your feedback. Let me start with one "platform" statement: Flex is the application server solution with a service-oriented client layer built on top of the Flash Player.

Now, I'll jump to the point where you started agreeing with us and from there will walk through the list of concerns all the way back.

4. "Flash has no threads, no thread management."

I find this rather hard to justify as it is.

Multithreading capabilities are implemented in browsers as well as in the Flash Player. You may question, however, the level at which these capabilities are available to a programmer. After all, the beauty of XMLHTTPRequest is that it is asynchronous, isn't it? Otherwise we'd be saying JAX instead of AJAX :). Similarly, the same asynchrony has been available with Flash Remoting since 2002 or 2003, if I am not mistaken.

Let's take your use case – "some calculation." Must be something CPU worthy, I guess. The question is where does it belong in the distributed system, regardless of the Swing/Flash debate. Perhaps on the server, closer to data sources? Then, using the remoting capability of Flex/Flash, I would suggest a POJO running not only in a separate thread, but also on a separate machine, across the wire!

Now, just out of curiosity, let's try to play without the server, with Flash alone. Here is another take: can you have another "servant" application run by the Flash Player within the same hosting HTML page? Can you interop via the LocalConnection object to invoke methods, pass parameters – all with complete marshalling of complex data types to native objects?

Wouldn't it be happening in a different thread?

Perhaps we can come to a more accurate statement: there is no pre-emptive multithreading within a single Flash VM. This might indeed be an issue if we had to take distributed computing out of the picture.

But we don't have to, do we?

3. "There are no skins for different components..."

This one is simpler. If you are, in fact, talking about Flex, which has a totally different code base than Flash controls, the statement is outright ungrounded. Flex controls support pretty advanced skinning, although my fascination with the subject went south after I skinned a couple of controls.

But then here's another part: "...Adobe...main interest is in popularizing visual effects..."

Well, this is one very popular illusion, I might say. How about this answer: Adobe Flex offers developers a JMS adapter that enables them to create a producer or consumer with one line of XML code? How _visual_ is that?

2. "Flash components are closed source..."

I have a secret to tell. Flex comes with full sources. Look at them, step them through, do whatever you please. Just don't tell Adobe I told you :). Seriously, are we sure we are talking about the same products here?

Our article was about Flex.

1. "Flash components are badly written..."

If indeed we are both talking about Flex, I find their components extremely well done. Not that I doubt for a split second that you can find a handful of cracks in each of them. But, being an expert, you naturally see how to avoid a problem – in another split second, don't you?

Also, now that you have the full source code of the controls (you do, I kid you not), what stops you from overriding any given method and creating your own Accordion or whatever? The Flex community and Flex engineers are very friendly people who will gladly accept and appreciate any good ideas you might want to offer.

Register by July 14
to save up to $500
Enter priority code D0120

15 Years of BREAKING the rules. . .

Now we're Finally **MAKING** the Rules!

SAN FRANCISCO'06

Join LinuxWorld as we celebrate the 15th Anniversary of Linux—and its amazing evolution from a kernel to industrial-strength corporate applications. Come to San Francisco—to the main event for the Linux community—and dive into the open technologies that are still transforming IT all these years later.

Go Deep in technical training
Brainstorm with the best
Check Out the latest products and services
Feed Your Mind in visionary keynotes
Feel the Energy of the entire Linux and open source community

Go Deep in 100+ Sessions on Key Linux and Open Source Topics

- Iron-Clad Security for Open Environments
- Managing Mixed Environments
- Virtualization
- Scalable Open Source Applications
- Desktop Linux
- Grid Computing
- Mobile Linux
- Web Services and SOA
- VoIP
- IT / Line-of-Business Alignment
- High-Performance Open Source
- The Business of Open Source
- Kernel and System Development

Register by July 14 and save
Enter priority code D0120 and save up to $500
www.linuxworldexpo.com/register

Brainstorm with Top Open Source Experts

- **Alan Boda,** Software Consultant, HP
- **Fabrizio Capobianco,** CEO, Funambol, Inc.
- **Dave Dargo,** Senior VP for Strategy and CTO, Ingres
- **Chris DiBona,** Open Source Programs Manager, Google
- **Scott Handy,** VP of Worldwide Linux & Open Source, IBM
- **Greg Kroah-Hartman,** Linux Kernel Subsystem Maintainer, SuSE Labs / Novell
- **Dan Kusnetzky,** Executive VP of Marketing, Open-Xchange Inc.
- **Eben Moglen,** Chairman, Software Freedom Law Center
- **Bernard Traversat,** Director of Advanced Development, Sun Microsystems

Don't Miss the Biggest Linux Event Ever!
Hundreds of products, services, and training sessions all under one roof!

KEYNOTE SPEAKERS

Lawrence Lessig	**Guru Vasudeva**	**Peter Levine**	**Greg Besio**	**Richard Wirt**
Professor of Law, Stanford Law School	*AVP & Enterprise Chief Architect, Nationwide*	*CEO, XenSource*	*Corporate VP, Mobile Devices Software, Motorola*	*VP, Senior Fellow, General Manager, Software and Solutions Group, Intel*

Check out the latest products and services from hundreds of key exhibitors

AMD	HP	Oracle
CA	IBM	Unisys
Dell	Intel	VMware
EMC	Novell	and more!

Feed Your Mind in Visionary Keynotes by Lawrence Lessig and other Linux and open source luminaries

Feel the Energy of a full slate of 15th Anniversary Events

Discover the Full Power of Today's Linux
Get the Skills, Solutions, and Insight Your Business Needs

Get complete details at www.LinuxWorldExpo.com and sign up for the event where Linux history is made.
Register by July 14 with priority code D0120 and save up to $500.

LINUXWORLD® CONFERENCE & EXPO

Conference: August 14 – 17, 2006
Expo: August 15 – 17, 2006

Moscone Center, San Francisco

Plus expanded content with OpenSolutions WORLD™

hp invent — PLATINUM SPONSOR | IBM — PLATINUM SPONSOR | MOTOROLA — PLATINUM SPONSOR | Novell — PLATINUM SPONSOR | ORACLE — PLATINUM SPONSOR | palmsource An ACCESS Company — PLATINUM SPONSOR | coverity — SILVER SPONSOR | SAP — SILVER SPONSOR | WYSE — SILVER SPONSOR | OpenSource — MEDIA SPONSOR

LinuxWorld is open to business professionals only. No one under 18 years of age will be admitted. D0120

IDG WORLD EXPO

JDJ News

Sun Adds Java DB and Swing Visual Designer to JDK and Enters Next Phase for Java Platform Standard Edition 6

(Santa Clara, CA) – Sun Microsystems, Inc., has announced it will be incorporating Java DB, the Sun supported distribution of the open source Apache Derby Project, as well as the Group Layout component from the NetBeans GUI Builder code-named Project Matisse (https://swing-layout.dev.java.net/) into the latest version of the Java(TM) Platform Standard Edition 6 (Java SE 6) Java Development Kit (JDK). In addition, Sun announced new agreements with Founder Technology Group and Lenovo to ship the Java Runtime Environment (JRE) on their hardware.

The second Beta release of Java SE 6 technology is now available at http://java.sun.com/javase/6 . Developers are encouraged to begin their transition to the Java SE 6 platform and leverage the enhancements and expanded functionality of the latest release. Scheduled for final release in the Fall of 2006, the Java SE 6 platform is the result of an industry-wide development effort that involves open review, weekly builds, and extensive collaboration between Sun engineers and over 330 external developers. In addition, Sun announced the expansion of service programs for Java SE 6 developers ranging from programming-specific advice to enterprise support with its Sun Developer Expert Assistance (DEA) Program and Sun Developer Service Plans (DSP).

QUALCOMM Java Solution Gains New Flexibility with Multitasking Capability

(San Diego) –- QUALCOMM Incorporated, a developer and innovator of Code Division Multiple Access (CDMA) and other advanced wireless technologies, has announced that select Mobile Station Modem (MSM) chipsets now support the concurrent execution of multiple Java applications. This multitasking capability extends the existing features of the QUALCOMM Virtual Machine (QVM) Java solution and delivers a seamless experience to wireless users simultaneously running multiple applications. The new multitasking capability is now available on the MSM6500 solution, and will be available on select additional chipsets thereafter.
http://www.qualcomm.com

BEA Announces WebLogic 9.2; Award-Winning Family Raises the Bar on SOA Enablement

(San Jose, CA) – BEA Systems, a provider of enterprise infrastructure software, has announced the general availability of WebLogic Portal 9.2, WebLogic Server 9.2, and BEA Workshop for WebLogic 9.2.

BEA WebLogic Portal 9.2 is a JEE-based enterprise portal server that is designed to help simplify the production and management of custom service-oriented portals. Among new tooling, federation, and community enhancements in WebLogic Portal 9.2 are new dynamic, adaptive user interface capabilities with rich granular features such as enhancements for AJAX support and market-leading support for Web Services for Remote Portlets (WSRP). Combined, the upgraded portal is designed to provide greater competitive advantage with increased flexibility to help adapt to business changes and richer more responsive user interfaces.
www.bea.com

Java Developer's Journal Advertiser Index

Advertiser	URL	Phone	Page
Quest	www.quest.com/JavaCode		2
Altova	www.altova.com	978-816-1600	4
IBM	ibm.com/takebackcontrol/flexible		7
Intersystems	www.InterSystems.com/Cache21P	617-621-0600	13
OPNET	www.opnet.com/pinpoint	240-497-3000	15
SAP TECHED	www.sapteched.com		17
Fiorano	www.fiorano.com/downloadsoa	800-663-3621	23
Cynergy	www.cynergysytems/thatsme		27
Backbase	www.backbase.com/jsf	866-800-8996	31
Tibco	www.tibco.com/mk/gi	800-420-8450	35
Instantiations	www.instantiations.com/rcpdeveloper	800-808-3737	43
RogueWave Software	www.roguewave.com/developer/downloads		47
Roaring Penguin	www.roaringpenguin.com	613-213-6599	49
Real World Flex Seminar	www.flexseminar.com	201-802-3020	53
AJAXWorld Conference & Expo	www.AjaxWorldExpo.com	201-802-3020	54, 55
Northwoods	www.nwoods.com	800-434-9820	57
LinuxWorld Conference & Expo	www.LinuxWorldExpo.com		61
SoftwareFX	www.softwarefx.com		63
Parasoft	www.parasoft.com/JDJmagazine	888-305-0041x3501	64

Eclipse Data Visualization

(No Silly Glasses Required)

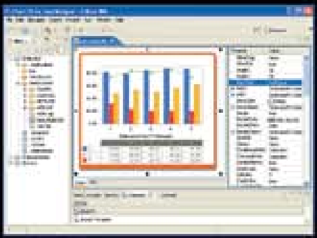


Chart FX for Java Eclipse plug-in.

The Leading Charting Solution Now Provides Powerful Data Visualization for Eclipse

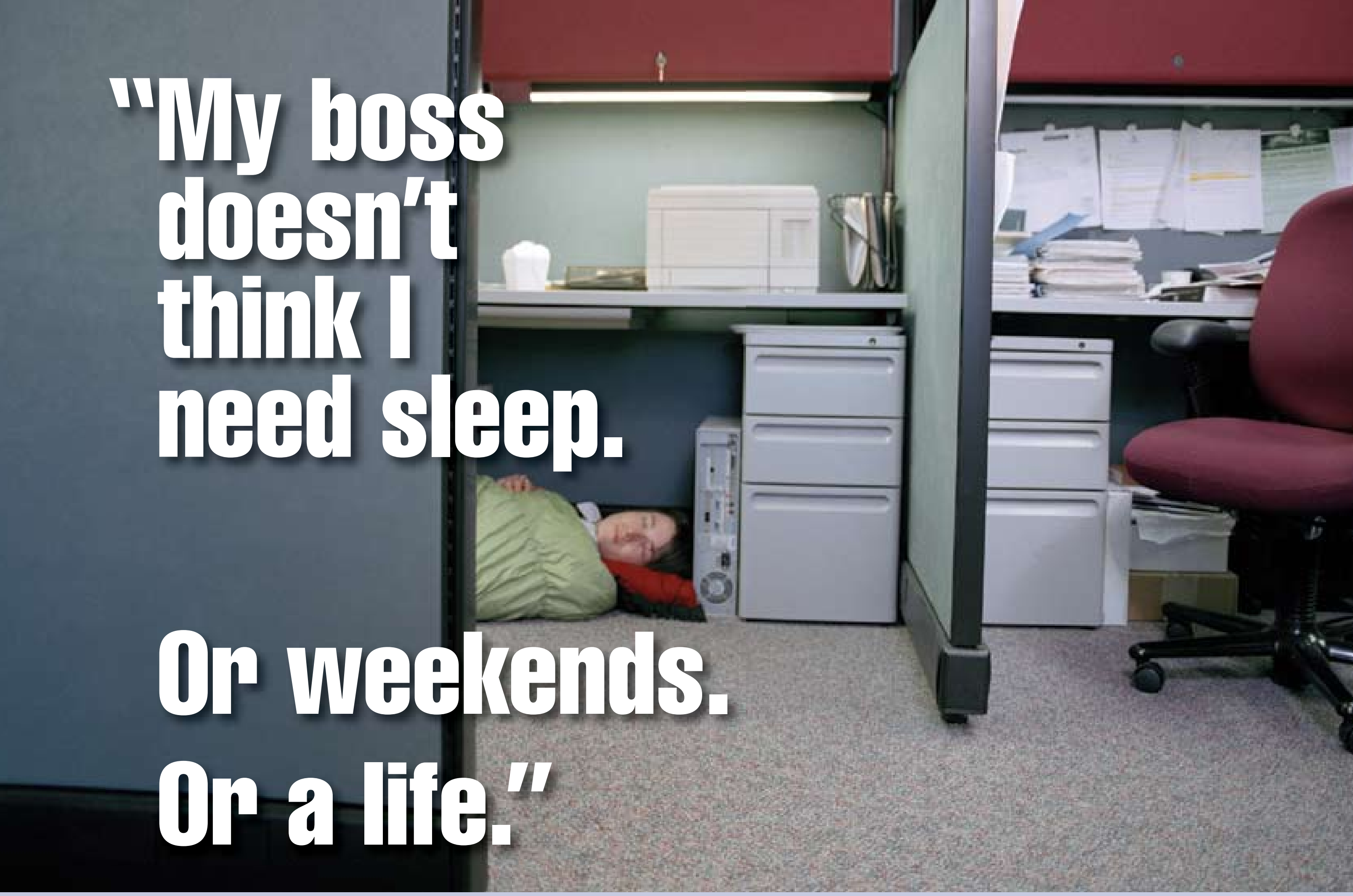
The **Chart FX for Java 6.2 Eclipse plug-in** brings enterprise-level data visualization features to the Eclipse IDE. The Designer is integrated into the IDE allowing quick customization of the charts and the required code generation. In addition to a myriad of traditional chart types, the **Chart FX Maps** extension is included to create dynamic, data-driven image maps, such as geographic maps, seating charts or network diagrams, among others. Chart FX for Java 6.2 is available as a Server-side Bean that runs on most popular Java Application Servers. The 100% Java component produces charts in PNG, JPEG, SVG and FLASH formats. The **Chart FX Resource Center** integrates into the Eclipse Help and includes a Programmer's Guide, the Javadoc API and hundreds of samples. This makes Chart FX for Java the most feature-rich, easy-to-use charting tool available for Java development. *Learn more about the seamless integration and powerful features at www.softwarefx.com.*

Chart FX

www.softwarefx.com

New! version 6.2

Now Includes Maps!



**If your delivery cycle is taking priority over REM cycles,
Jtest's Automated Error Prevention brings your schedule back to reality.**

Sleeping in your cubicle is no way to stay on schedule. You need to test while you work, not after everything is done. With Jtest, you'll be home in time for dinner. And a little rest.

Jtest automates the valuable yet tedious task of code reviews and unit testing, allowing development teams to adopt a "test as you go" strategy. This promotes the testing of code as it's developed, so that **quality is built into the code from its inception.** And bugs are eliminated before they infect the rest of the code base.

The result is cleaner, safer, more reliable and consistent code. You'll be reworking less code, which means faster and more predictable release cycles. And more sleep.

- Jtest checks whether code complies with 500+ Java development rules and automatically corrects many reported violations.
- Automatically unit-tests your Java code, evaluating code behavior and exposing unexpected exceptions, boundary condition errors and memory leaks.
- Generates extendable, high-coverage test cases in JUnit format, which can be quickly turned into libraries of reusable test assets.
- Improves productivity by helping team members leverage one another's test settings and related files.

PARASOFT®
We make software work.™

Download your FREE TRIAL at www.parasoft.com/JDJmagazine Or call (888) 305-0041, x3501