LOOKING BACK, LOOKING AHEAD PAGE 62

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

JDJ

JDJ.SYS-CON.COM VOL.11 ISSUE:1

IN THIS ISSUE...

JSF and AJAX

INTRODUCING A NEW OPEN SOURCE PROJECT

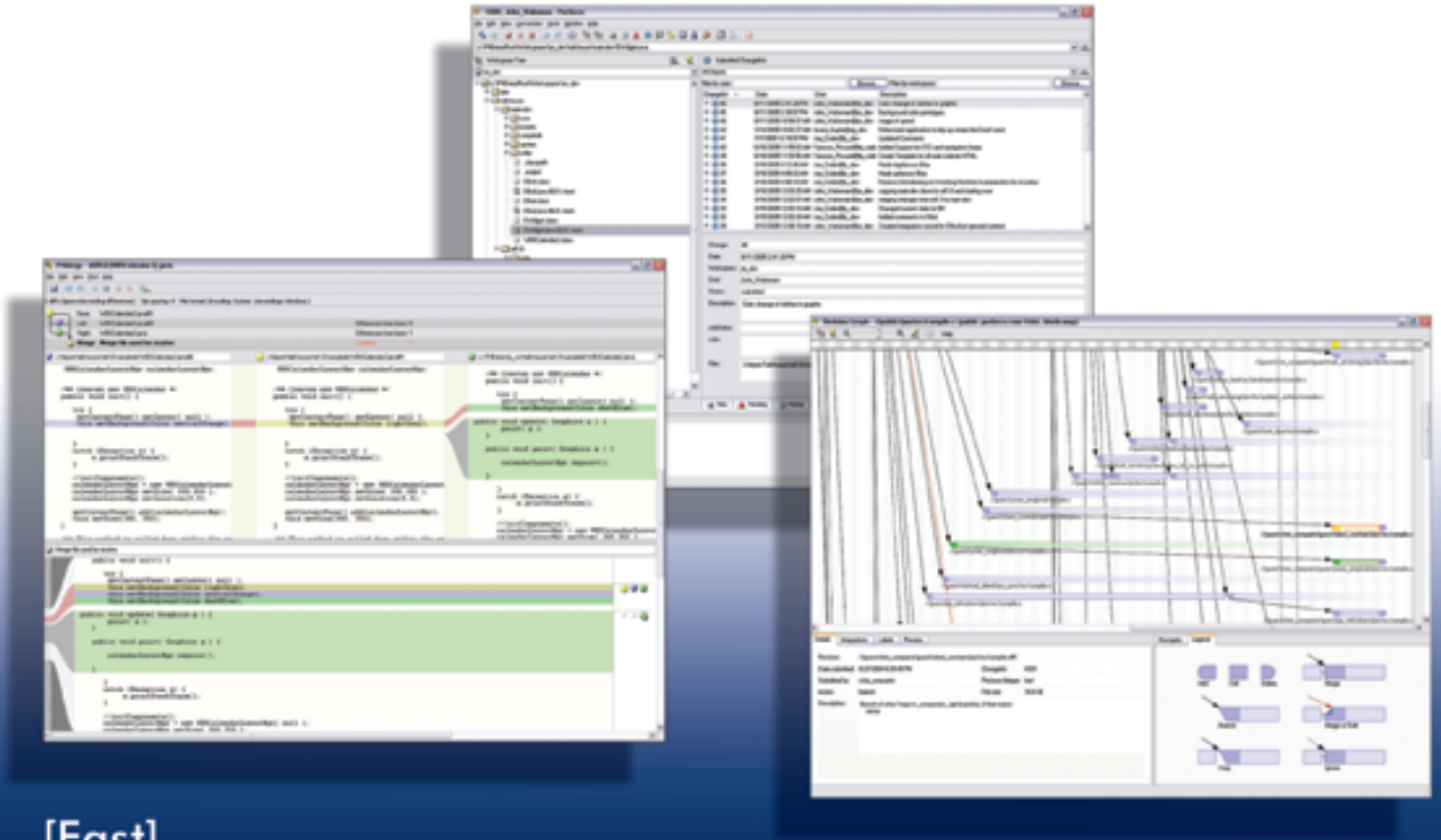
RETAILERS PLEASE DISPLAY UNTIL MARCH 31, 2006
$5.99US $6.99CAN
02
0 09281 01751 6

PLUS...

- Reporting Made Easy
- Struts Validations Framework Using AJAX
- Introduction to Acegi
- Proxy Cache

Perforce Software Configuration Management

Powerful and scalable.



[Fast]

[Scalable]

[Distributed]

Perforce tracks and manages source code and digital asset development, giving you maximum control with minimum interference.

Perforce scales to **thousands of concurrent users.** It manages code bases of **more than a million files.** It processes **terabytes of data** - from source code to documents, web content, and image files.

Perforce is fast, industrial-strength SCM. Any team size, any amount of data - Perforce handles it.

PERFORCE SOFTWARE

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

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
Production Consultant: **Jim Morgan**
Associate Art Director: **Tami Lima**
Executive Editor: **Nancy Valentine**
Associate Editor: **Seta Papazian**
Research Editor: **Bahadir Karuv, PhD**

Writers in This Issue
Man-ping Grace Chau, Yakov Fain, John Fallows, Jess Garms, Jeremy Geelan, Tim Hanson, David Hardwick, Sonny Hastomo, Jonas Jacobi, Ganesh Kirti, Onno Kluyt, Justin Knowlden, Raymond K. Ng, Peter Sellars, Joe Winchester

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.

Worldwide Newsstand Distribution
Curtis Circulation Company, New Milford, NJ
For List Rental Information:
Kevin Collopy: 845 731-2684, kevin.collopy@edithroman.com
Frank Cipolla: 845 731-3832, frank.cipolla@epostdirect.com

Newsstand Distribution Consultant
Brian J. Gregory/Gregory Associates/W.R.D.S.
732 607-9941, BJGAssociates@cs.com

The Shape of *i*-Technology to Come

Jeremy Geelan

This is traditionally the time of year for SYS-CON Media's roundup of i-Technology predictions from around the Web and the year's harvest of thoughts and viewpoints. According to our worldwide network of software development activists, evangelists, and executives, 2006 promises to be a vintage year for software development…

Take Microsoft, for example: A new client OS is on the way, Microsoft Vista, due late in 2006, giving rise to the obvious question: Will the new cool 3D user interface be enough to move the user to upgrade? We'll see. Maybe the new built-in security, performance features, and integrated search will be enough to convince users – after all, why go to the Web if built-in Web-enabled services and integrated information search allow the Web to come to you?

Or consider the world of PDA devices. Everyone is looking for the next killer Palm or BlackBerry. But are they looking in the right direction for the next killer PDA? What about unexpected places – for example, Nintendo? Check out the new Nintendo DS: Could you imagine it running Pocket PC or Palm OS? That would make a very cool gadget. What about the iPod? Have you seen the new iTunes-enabled Cingular Phone? It could be closer than you think.

On the pages that follow you'll find the collected wisdom of some of the most acute prognosticators in the industry. As always with *JDJ* and SYS-CON Media, we don't ask pundits and sideline commentators but activists, folks whose connection with software development and/or the software industry is daily, intense, and driven by real-world concerns of ROI and the business case for innovation, not just innovation for innovation's sake.

As ever, please don't hesitate send us your own thoughts. "None of us is as smart as all of us," they say, a philosophy that has even spawned a book (*The Wisdom of Crowds: Why the Many Are Smarter Than the Few and How Collective Wisdom Shapes Business, Economies, Societies and Nations* by James Surowiecki,). We will publish a roundup of readers' predictions in the February issue of *Java Developer's Journal*.

Let's begin this year's roundup with the predictions for 2006 of **Mitchell Kertzman**, now at Hummer Winblad Venture Partners but still famous for having been the founder and CEO of Powersoft, which merged with Sybase in February 1995. When someone with over 30 years of experience as a CEO of public and private software companies tips LAMP, for example, it lends a certain credence to an already strong trend that we have sought to cover in SYS-CON Media's various publications such as *LinuxWorld Magazine* and over at OpenSourceEnterprise.com.

MITCHELL KERTZMAN
AJAX, LAMP, Virtualization, SaaS, Open Source

Since I'm in venture capital now, I try to put my (and others') money where my mouth is, so my predictions will tend to match up with my portfolio.

In no particular order:

1. Rich application interfaces, including (but not exclusively) AJAX. Enterprise developers/IT managers have finally realized that the browser interface was a step backward to the 3270 and forms mode. That was good enough for a while, but not anymore.
2. LAMP in the enterprise. If you follow my portfolio company, ActiveGrid, you'll find one of the leaders of the J2EE app server market now offering a far easier-to-build and less-expensive-to-deploy platform.
3. Virtualization. With three strong virtualization platforms (VMWare, Microsoft Virtual Server, and XenSource) now available, there will be more and more software products built not on traditional hardware/software platforms but on virtualized platforms. Check out Akimbi Systems, which provides a very exciting application for QA and testing in the enterprise.

–continued on page 60

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

jeremy@sys-con.com

Altova® MapForce® 2006 – The premier data integration and Web services implementation tool.

ALTOVA® mapforce® 2006

Streamline Web services

Hook up with MapForce® 2006, and build Web services without writing any code.

New in MapForce 2006:

- Drag-and-drop Web services implementation
- Advanced flat file parsing and integration
- Project-wide code generation
- Embedding in your applications via OLE / ActiveX

Altova® MapForce, the tool awarded for easy integration of XML, database, text, and EDI file formats, now also lets you implement Web services in a visual way. Simply drag connecting lines from information sources to targets and drop in data processing functions. MapForce converts data on-the-fly and auto-generates data mapping code in XSLT 1.0/2.0, XQuery, Java, C++, or C# for use in your data integration and Web services applications. Give your data direction!

Download MapForce® 2006 today: www.altova.com

Also available in the Altova XML Suite.

Supports Nov. '05 W3C Candidates for XSLT 2.0, XPath 2.0 & XQuery!

ALTOVA®

JANUARY 2006 VOLUME:11 ISSUE:1

JDJ contents

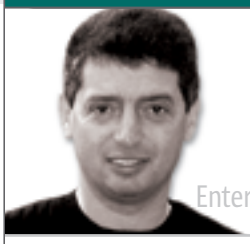
JDJ Cover Story

Features

Experiences with the New 1.5 Java Language Features
Jess Garms and Tim Hanson

The Flexible Model
by Man-ping Grace Chau

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.



Yakov Fain
Enterprise Editor

Java Champions

Recently I got an e-mail with the following header: "Your nomination to Sun Java Champions."

My Java-intoxicated brain immediately started several parallel threads. Since I now use the Callable interface instead of Runnable, my threads can return results and throw exceptions.

The first thread threw a SpamException.

The second thread returned a String "but there is no attachments."

The third one gave me "The sender's address ends with sun.com."

But I liked the last thread the most: "Just open the e-mail, will you!"

I did, and this is how the e-mail started: "I am contacting to let you know that you have been nominated to the Sun Java Champions Program through the Java Champions homepage https://java-champions.dev.java.net/.

The Java Champions program is sponsored by Sun Microsystems and is an effort to recognize leaders in the Java Community and invite them to participate in the development of the Java platform in collaboration with Sun engineers and Java Luminaries."

No kidding! I accepted this nomination immediately. Then several members of Java Champions Selection committee approved my nomination too, and here I am a Java Champion!

I'm really proud to be listed on the same Web page with James Gosling, Doug Lea, Bill Venners, Gavin King, and other great Java world leaders. This program seems to be similar to Microsoft's Most Valuable Professional (MVP) deal, but let me find out from the source. Matt Thompson, director of Sun Developer Network & Open Source Programs Office has agreed to answer my questions.

Q: When was the Sun Java Champions program introduced?

A: It was introduced in June 2005 at the JavaOne conference in San Francisco.

Q: How do you decide who will get the T-shirt with the logo "Sun Java Champion"?

– continued on page 8

> "The Java Champions program is sponsored by Sun Microsystems and is an effort to recognize leaders in the Java Community"

Yakov Fain is a senior technical architect at BusinessEdge Solutions, a large consulting and integration firm. He authored the best selling book "The Java Tutorial for the Real World," an e-book "Java Programming for Kids, Parents and Grandparents," (smartdataprocessing.com) and several chapters for "Java 2 Enterprise Edition 1.4 Bible." He leads Princeton Java Users Group.

yakovfain@sys-con.com

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*
Advertising Sales Director:
Robyn Forma *robyn@sys-con.com*
National Sales and Marketing Manager:
Dennis Leavey *dennis@sys-con.com*
Advertising Sales Manager:
Megan Mussa *megan@sys-con.com*
Associate Sales Manager:
Kerry Mealia *kerry@sys-con.com*

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

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

Web Services
Information Systems Consultant:
Robert Diamond *robert@sys-con.com*
Web Designer:
Stephen Kilmurray *stephen@sys-con.com*

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

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

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

Innovations by InterSystems

Rapid development with robust objects

Lightning speed with a multidimensional engine

Easy database administration

Massive scalability on minimal hardware

Real-Time Data Analytics With A Real-Fast Database.

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

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

InterSystems CACHÉ™

Try an innovative database for free: Download a fully functional, non-expiring copy of Caché, or request it on CD, at **www.InterSystems.com/Cache4P**

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

– continued from page 6

A: When we started the process, we looked around within the Java Community for leaders in the various parts of the Java eco-system (like JUG leaders, authors, trainers, professors, researchers, etc.). From this has grown a "community nomination" process that's both unique and a real strength. The community itself nominates most nominees and it selects new Java Champions through a peer review process. The criteria used are listed in the Java Champions homepage.

Q: How many Java Champions are there?

A: Currently there are about 70 Java Champions around the world, with another 40-50 in the selection process queue. Once we get up to 100 or so we're going to evaluate how many more are appropriate. The idea was always to make sure we built a community of Java Champions that reflects the top echelon of contributors to the Java Community.

Q: What feedback/contributions do you expect from the Champions?

A: I think what's interesting isn't what we expected, but rather what has happened. We've already engaged the Champions in a number of discussions about the state of Java training, the use of the Java brand, the state of Sun's Java tools, and several others. The overall discussion has been incredibly valuable and the feedback has been very well received within Sun.

As we go forward, we'll continue to share ideas and concepts with the Champions and use them as a sounding board for the larger Java community. The various discussions around the state of the Java platform are also an excellent resource for Sun to "take the pulse" of the Java eco-system. We're very fortunate to have a great group of Champions who are willing to share their views of how we can work together to continue to grow the adoption of the Java platform honestly and openly.

Q: I've already gotten a gift box of T-shirts, computer widgets, and a book. What other perks are you planning to offer to the |Champions?

A: There are a number of benefits besides the "goodies" that the Champions have already received. One is the opportunity to meet directly with Sun leadership to discuss issues and opportunities around the Java platform. One of these meetings just took place at JavaPolis in Antwerp, where Jeff Jackson, senior vice-president of the Java platform, tools, and Sun's developer program (the Sun Developer Network), sat down with several Champions to discuss the state of the platform.

In the future we'll look at different ways to work directly with individual Champions to potentially highlight their work (either through speaking opportunities at Sun's developer events like JavaOne or the Sun Tech Days or online through Sun's developer portal: developers.sun.com).

Q: Now on the lighter note do you think a Java Champion can be called "The Brother or Sister of Java" or at least "The Cousin of Java"?

A: I look at these folks as the heroes of the Java platform – so brother/sister/cousin doesn't imply enough value (at least to me). These folks are truly both a wealth of knowledge for us to tap into, as well as a great resource to work with in making the Java platform easier to adopt worldwide.

Yakov. Thank you, Matt!

I'm very down-to-earth and realize that even though I'm a Champion, you can find plenty of people who know Java better than me. But you won't find too many people who enjoy living in this great Universe called Java Community more than I do.

> "The various discussions around the state of the Java platform are also an excellent resource for Sun to "take the pulse" of the Java ecosystem"

THE MOST CUTTING EDGE DEVELOPMENT IN REPORTING SOFTWARE GIVES USERS SOME MUCH-DESIRED FREEDOM.

THE ONLY REPORTING SOFTWARE THAT LETS USERS DESIGN THEIR OWN REPORTS IN MSWORD. YES, REALLY.

Free Yourself from the Task of Report Design

At last! There's a way for developers to rid themselves of the daunting and time consuming task of trying to design report templates that satisfy business owners and managers. That's because Windward Reports enables them to design their own reports using popular, easy-to-use word processing programs like Microsoft Word – so there is no new software to learn.

Point. Click. Done.

Windward Reports utilizes Microsoft Word, the word processing software program that virtually everyone knows (and frankly, it works with just about every other word processing program, as well.) All a business owner needs to do is open up Word's deep, rich library of design templates and configure the design that they want. So designing a report is now as easy as creating any MSWord document. Your data just flows in and completes the picture, creating reports that will dazzle and impress. It's really that easy.

Find Out for Yourself with a Free, No-Obligation Demo.
Go to www.windwardreports.com. You won't believe your eyes.

WindwardReports
advanced J2EE/.NET report generation

Reporting Made Easy

JasperReports and Hibernate in Web applications

by Peter Sellars

JasperReports is a valuable and viable reporting solution for Java Web applications. It simplifies report generation through the use of XML report templates that are then compiled using the JasperReports engine for use in reporting modules. These compiled report templates can be filled by data received from a variety of sources including relational databases. JasperReports can be integrated into Web applications and create reports in several file formats including PDF and XLS.

Reporting in Java Applications

Often reporting modules increase in complexity and size during the course of application development. Clients tend to demand more information from report modules when they become aware of the benefits reports offer. The reporting module developed as something of an afterthought in such environments suddenly becomes a much more integral part of the application. Reporting modules often seem to be tacked on to developed applications, rather than being considered and implemented during initial application development.

Recently while working on some applications that made extensive use of report extraction to XLS files using the Apache POI library, it became apparent that these report modules tied up lots of valuable development resources for extended periods of time. When the client requested PDF extraction, initial iText API research led me to discover JasperReports. JasperReports was to change our team approach to report development dramatically.

Prior to implementing JasperReports each report creation required the development of a custom report class using the Apache POI library. This approach expended valuable development time creating aspects of the report such as cell specific formats, styles, and population methods. JasperReports offered our team the ability to get back this valuable development time, while producing the same report because of its embedded use of the Apache POI library.

One of the benefits offered by the introduction of JasperReports is that a single report template implementation can produce reports in a number of formats. This means that templates created for XLS format extraction can also be used to produce PDF files and even CSV, HTML or XML.

How Can JasperReports Help Developers?

JasperReports gives developers the ability to create reports quickly and easily that can be extracted to numerous formats. Developers can also use the JasperReports engine to compile report templates at design or runtime – allowing dynamic report formats. Developers can also inject data into these reports from a number of data sources. Developer time no longer has to be spent creating custom report classes using the Apache POI or iText libraries for formatting and stylizing reports, allowing the code writers to focus on the data retrieval aspect of the report. As a result developers gain valuable flexibility and time savings using JasperReports in application development.

The XML report templates used by JasperReports provide the layout and presentation information required to format the resulting report as well as field, variable, and parameter references. Non-development staff can create these templates using a third-party GUI such as iReport with minimal developer collaboration, so developers don't have to involve themselves in the layout and presentation aspect of report generation.

JasperReports enables developers to concentrate their efforts on the parts of the reporting module where they are required, while relieving them of having to write custom report generation code. A developer's role in the report module can be reduced to template compilation, data source implementation, and actual report creation.

Creating and Compiling an XML Report Template

JasperReports requires a report design defining the layout, presentation, and data fields. This design can be built using the *net.sf.jasperreports.engine.design.JasperDesign* object, so developers can create report designs dynamically, or by creating a *net.sf.jasperreports.engine.design.JasperDesign* instance from an XML report template. Unless an application specifically requires a dynamic layout a compiled XML report template is the recommended method. This XML template is usually saved with a .jrxml file extension and compiled using the *net.sf.jasperreports.engine.JasperCompileManager*.

The JasperReports XML template includes elements for <title>, <pageHeader>, <columnHeader>, <pageFooter>, <columnFooter>, and the main data <detail> element. Each of these elements has a variety of sub-elements as can be seen in *sampleReport.jrxml* (see Listing 1).

You can download the code samples used in this article at jdj.sys-con.com. As can be seen in *sampleReport.jrxml*

Peter Sellars is a Java developer who develops Web applications.

peter@netbyte.com

some elements such as <band> and <reportElement> contain layout information, while others such as <textElement> and <font> contain presentation information. The XML templates also contain <parameter>, <field>, and <variable> elements used to include data in the report.

The <parameter> elements allow non-data source information to be passed into a report, such as a dynamic title; <field> elements are the only way to map report fields to the data source fields, while variables are values generated at runtime for use in the report. The complete Document Type Definition (DTD) for the JasperReports XML report template can be found in the JasperReports Ultimate Guide.

Compilation of the XML template can be done either at runtime or build time as part of an Ant build using the JasperReports Ant task.

Compiling the report at runtime entails loading the report into a *JasperDesign* object and using the created instance as the parameter to the *JasperCompileManager.compileReport(JasperDesign design)* method, which returns a *JasperReport* instance. Alternatively the XML template can be passed into the *JasperCompileManager.compileToFileReport(String sourceFileName,* which creates a compiled report file (.jasper) available throughout the application.

Compiling the report at build time using the JasperReports Ant task requires the addition of the task definition to the build.xml file and a target making use of this task as seen in Listing 2, which is an extract from the source code build.xml. Using the Ant task results in the creation of a compiled (.jasper) file in the *destdir* task and offers the opportunity to save the Java source file by passing the *keepjava* attribute of the target a true value. A more thorough example of how to use the Ant task is included in the sample applications provided in the JasperReports download bundle.

Using Data Sources to Fill JasperReports

Most reports use a database as the data source, but JasperReports can use any available data source. These data sources are passed to a *net.sf.jasperreports.engine.JasperFillManager fillReportXXX()* method. Two types of data source are provided for by these methods – *net.sf.jasperreports.engine.JRDataSource* and *java.sql.Connection*. The source code for this article contains examples of both a static data source that extends the *JRDataSource* and a JDBC connection data source implementation.

The *StaticDataSource* class implementation provided implements the *net.sf.jasperreports.engine.JRDataSource* interface enabling it to fill the report data by calling the *JasperFillManager.fillReport(JaperReport report, Map parameters, JRDataSource dataSource)* method. The two required methods *getFieldValue(JRField jrField)* and *next()* of the *JRDataSource* interface present in *StaticDataSource* handle the data passing from the data source into the JasperReport. The data source used by *StaticDataSource* is a static simple two-dimensional array of bowlers containing their names and scores over three games (see Listing 3). When the *fillReport()* method containing this data source is processed and a detail section is encountered in the report a call will be made to the *next()* method. The implementation of this method in *StaticDataSource* (see Listing 4) returns true if there's another element in the data array, or false if there is no more data. If this method returns true then field elements encountered in the detail section will result in a call to the *getFieldValue(JRField jrField)* method in *StaticDataSource*. The implementation of this method in *StaticDataSource* (see Listing 5) returns the value of the mapped data field name for the current index of the data array. When the end of the detail section is encountered, the *next()* method is called again and the process repeats until the *next()* method returns false.

The *JDBCDataSourceExample* (see Listing 6) implements a *fillReport()* method that accepts a *java.sql.Connection* parameter. Through the addition of a <queryString> element into the XML report template (*jdbcSampleReport.jrxml*) this *fillReport()* method enables data to be extracted from a relational database. The <queryString> element returns the data fields for use in the report data mapping. In this case the query simply returns all records in the sample_data table. A *java.sql.ResultSet* can be used instead of implementing the <queryString> element in the report template, allowing dynamic query implementation.

Using Hibernate with JasperReports

Hibernate is one of the most popular ORM tools in use at the moment. Using Hibernate as a data source for JasperReports can be very simple when a collection of objects is returned from a Hibernate query, but when a tuple of objects is returned then a custom *JRDataSource* implementation is required.

When a Hibernate query returns a collection of objects, a *net.sf.jasperreports.engine.data.JRBeanCollectionDataSource* can be used to map the Hibernate POJO instance fields to the report fields. All that's required for this simple solution is to use the *JRBeanCollectionDataSource(java.util.Collection beanCollection)* constructor, passing it the Hibernate Query result set as implemented in *SimpleHibernateExample* (see Listing 7). In this example the simple Hibernate query used (*session.createQuery("from SampleData").list()*) is equivalent to that found in the *JDBCDataSourceExample*. *JRBeanCollectionDataSource* implements *JRDataSource* like *StaticDataSource* but its *getFieldValue(JRField jrField)* method implementation maps the report template field names to the query result bean properties.

When a Hibernate query returns a tuple of objects it's necessary to write a custom implementation of the *JRDataSource* similar to *HibernateDataSource* (see Listing 8). The implementation of the required *next()* method in this class returns true if there is another list item in the Hibernate query result set, while putting the current list item in a *currentValue* holder for use in the *getFieldValue(JRField jrField)* method. The *getFieldValue()* method implementation gets the field index in the *currentValue* object via a call to the *getFieldIndex(String field)* method. This method iterates through the mapped field names passed to the *HibernateDataSource* constructor until it finds the field name it was passed and then returns the index of this field in the currentValue information. The *getFieldValue()* method then returns the value at this index in the currentValue result object.

More extensive solutions to using Hibernate with JasperReports, including the use of reflection instead of the name mapping method used in *HibernateDataSource*, can be found on the Hibernate Web site *http://www.hibernate.org/79.html*. Also of interest in this area is the report optimization implementation advocated by John Ferguson Smart in his article "*Hibernate Querying 103: Using Hibernate Queries with JasperReports*" (see Resources).

Exporting Reports to PDF and XLS Formats in Web Applications

After compiling and filling a JasperReport report exporting it is a fairly simple and straightforward process using the *net.sf.jasperreports.engine.JRExporter* interface implementations provided. JasperReports can export data to PDF, XLS, CSV, RTF, HTML, and XML from the same report design using the appropriate implementation of the JRExporter interface. The PDF and XLS formats are two of the most common export formats and examples of exporting to these formats from within a Web application can be found in the source code for this article. *PrintServlet* exports to PDF, while *DataExtractServlet* exports the same data to an XLS format file.

PrintServlet (see Listing 9) is a an example servlet implementation class using JasperReports to export a report to PDF format. JasperReports makes use of the Open Source iText PDF creation library (see Resources) to generate PDF format files. Once the report is compiled in *PrintServlet,* the PDF is created and streamed to the Web browser ready for printing using the *runReportToPdfStream(InputStream inputStream, OutputStream outputStream, Parameters params, Connection connection)* method implemented by the *JasperRunManager* facade class.

DataExtractServlet (see Listing 10) is an example servlet implementation class using JasperReports to export a report to the XLS format. JasperReports makes use of the Apache POI library (see Resources) to generate XLS format files. Once the report is compiled in *DataExtractServlet* the XLS file is created in memory and a save dialog is displayed to the user. The servlet uses *net.sf.jasperReports.engine.export.JRXlsExporter,* one of the concrete implementations of the *JRExporter* interface provided by JasperReports to export the report. The parameters for exporting the report are initialized using *JRXlsExporterParameter* variables to set the filled report (*JRXlsExporterParameter.JASPER_PRINT*) and the output stream (*JRXlsExporterParameter.OUTPUT_STREAM*) – which is the response object that has had its content type and header set so that the file will be made available to the user for saving rather than displayed as in the PrintServlet example when *exportReport()* is called.

Useful Hint: By default JasperReport puts page headings at the top of every 'page' of data. When exporting to an XLS format this breaks up the continuous data in a worksheet that contains more than a single 'page' of data. Data continuity can be maintained by passing the type of output format as a parameter to a report template combined with a <*printExpression*> element based on the passed parameter placed in the <*pageHeader*> element. The <*printExpression*> below will result in only the page headings being output to a 'page' if the report is processing the first page when the output format isn't PDF and on every 'page' for PDF output formats.

```
<printWhenExpression>
 <![CDATA[$V{PAGE_NUMBER}.intValue() == 1
   || $P{REPORT_TYPE}.equals("PDF")
    ? Boolean.TRUE : Boolean.FALSE]]>
</printWhenExpression>
```

Creating Reports Is Easy and Fun with JasperReports

Hopefully this article has whetted your appetite for exploring the world of report generation using JasperReports, or if you've already discovered JasperReports, that it's provided some ideas on how to delve into creating custom data sources or using new export formats. Understanding and mastering the implementation of the required *JRDataSource* methods *next()* and *getFieldValue(JRField jrField)* opens up any data source for use in generating reports with JasperReports.

Creating reports with JasperReports is made even simpler by some useful tools. iReport (see Resources), an excellent JasperReports template creation tool that allows visual report designs in a GUI application can be used by non-developers to create the JasperReport designs. It also offers substantial developer-focused functionality such as data source connectivity to create report previews outside of an application. JasperAssistant (see Resources), while not Open Source has the advantage of being an Eclipse plug-in for developing JasperReport templates in a similar GUI manner, albeit more developer-oriented. Both offer the benefit of being able to prepare a report design, which can then be provided to a developer for filling, relieving him of the tedious presentation aspect of report generation.

This article has barely scratched the surface of JasperReports' extensive use and functionality but hopefully it's introduced some developers to an extremely useful tool in any Java developer's arsenal. JasperReports can even produce charts and graphs, as well as including images in reports that increase the richness and presentation of an applications reporting system. JasperReports is a powerful API that can take a reporting system to the next level.

> "JasperReports dramatically changed our team approach to report development"

Resources

- JasperReports Ultimate Guide (Version 2.0), Teodor Danciu, JasperSoft Corporation.
- JasperReports Web site - http://jasperreports.sourceforge.net
- *i*Report Website - http://ireport.sourceforge.net
- JasperAssistant Web site - http://www.jasperassistant.com
- Hibernate Web site - http://www.hibernate.org
- Using JasperReports with Hibernate - http://www.hibernate.org/79.html
- Hibernate Querying 103: Building Reports with JasperReports - http://www.javalobby.org/articles/hibernatequery103/?source=archives
- *i*Text Website - http://www.lowagie.com/iText/
- Apache Jakarta POI Web site - http://jakarta.apache.org/poi/

Is your organization's content protected…

…and still easy to access and share?

Over two million users depend on Xythos to securely manage their content.

- Secure, web-enabled access to documents and files
- Easy-to-use document collaboration and workflow
- Integrated document classification and retention
- Rapid application and portal integration
- Proven reliability and scalability

Discover how Northeastern University became a CIO Bold 100 and Computerworld award winner with Xythos – Read the case study at www.xythos.com/neu

For more information please call 1.888.4XYTHOS or visit www.xythos.com
Xythos Software, Inc., 655 Montgomery, Suite 1600, San Francisco, CA 94111

Xythos
software, inc.

Struts Validations Framework Using AJAX

by Sonny Hastomo

Enriching the existing framework

Real-time data validation is one of the advantages of AJAX technology. By applying this technology, the struts validation framework will enrich the struts MVC and move the Web application closer to the desktop application.

The validation framework is used to validate fields. There are many ways to do validation on a Web application. It falls into two categories: server-side and client-side. A struts validation framework is one of the best frameworks for a Java-based Web application environment. It can configure the application using server-side validation and employ the error message that renders on the validation process invoked during the request processing time, or it can do client-side validation by using the JavaScript rendered on the requested page.

AJAX is a JavaScript technology that can asynchronously call the server and fetch the XML documents that are so popular lately. One of its uses is real-time data validation.

This article is concerned with enriching the existing struts validation framework with AJAX. A few components, such as a controller, have to be developed to select the validation framework and render the specific format message for the client side and a taglib to handle the error message rendering.

Sonny Hastomo is an IT architect at Jatis Solution company (Oracle and WebLogic Business Partner) for the product development division. His current focus is to integrate the J2EE Web framework, which is suitable for the product development, and interfacing other legacy applications based on the Unix or OS/400 Platform.

hastomo_net@yahoo.com

Prerequisites

You'll need a Windows system with Eclipse and the Tomcat application server. Make sure that the MSXML 3.0 ActiveX object is registered on your operating system. You'll also need the Struts library (http://struts.apache.

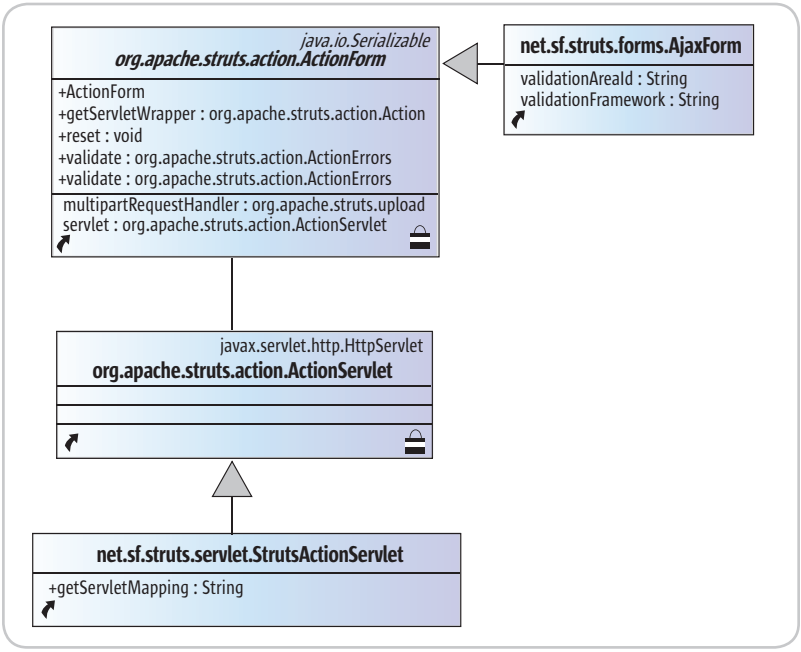


Figure 1 AjaxForm Class Hierarchy

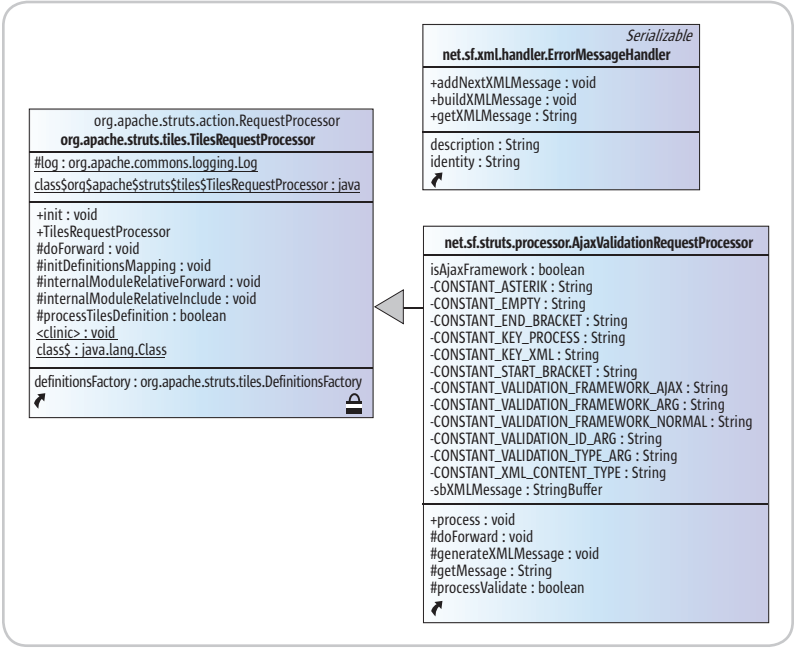


Figure 2 ErrorMessageHandler Class Hierarchy

org) and the JDOM library (http://www.jdom.org) for XML development (see Figures 1 and 2).

Server-Side Scenario

StrutsActionServlet

We have to extend the class from org.apache.struts.action.ActionServlet to get the servletMapping variable that stores the information on how the extension will be formatted for action classes into the action path as a browser address. When the code is added, we have to configure web.xml as a Web application descriptor for the application server.

The web.xml configuration:

```
…
<servlet>
<servlet-name>action</servlet-name>
<servlet-class>net.sf.struts.servlet.
StrutsActionServlet</servlet-class>
…
<servlet-mapping>
<servlet-name>action</servlet-name>
<url-pattern>*.do</url-pattern>
</servlet-mapping>
…
```

The StrutsActionServlet Java code looks like this:

```
public class StrutsActionServlet extends
the ActionServlet.
{
public String getServletMapping() {
return this.servletMapping;
}
}
```

AjaxValidationRequestProcessor

To support the existing Struts framework in the first step, we have to extend the RequestProcessor from the Struts package. We have to customize the request processor because we have to distinguish how we're going to do the validation – by using the existing Struts framework or the AJAX concept – and because we'll be making a contract between the server and client on how to interpret the message. In the message rendering we'll use the XML format, which is a good media messaging format. The definition of the XML format we'll apply is:

XML Format

```
<?xml version="1.0" encoding="UTF-8"?>
<message>
<identity name=messageAreaId>
            <description>
        MessageValue
</description>
</identity>
</message>
```

Description

- *Identity* is the ID for the client JavaScript to acknowledge where the message will be placed.
- *Description* is the result after the error message rendering from the server side.

First we need to get the servlet mapping configuration from the Web descriptor before moving onto the process mapping. After invoking the process, the application will prepare the instance of the form that inherits from the AjaxForm class. The processing manages AJAX validation and should check to ensure that the request from the client isn't using the struts validation framework. Other processes to perform during the request are process populate for collecting the information sent by the client into the action form, and process validation by using the method from the existing Struts validation framework that already exists in the parent class of AjaxValidationRequestProcessor (*TilesRequestProcessor*).

The validation process from the TilesRequestProcessor will invoke all the validation based on the struts validation framework and store the action errors into the request. What we need is to parse the action errors into pieces and generate the XML message validation that will be sent to the client side. Since we want to change the behavior of how the vali-

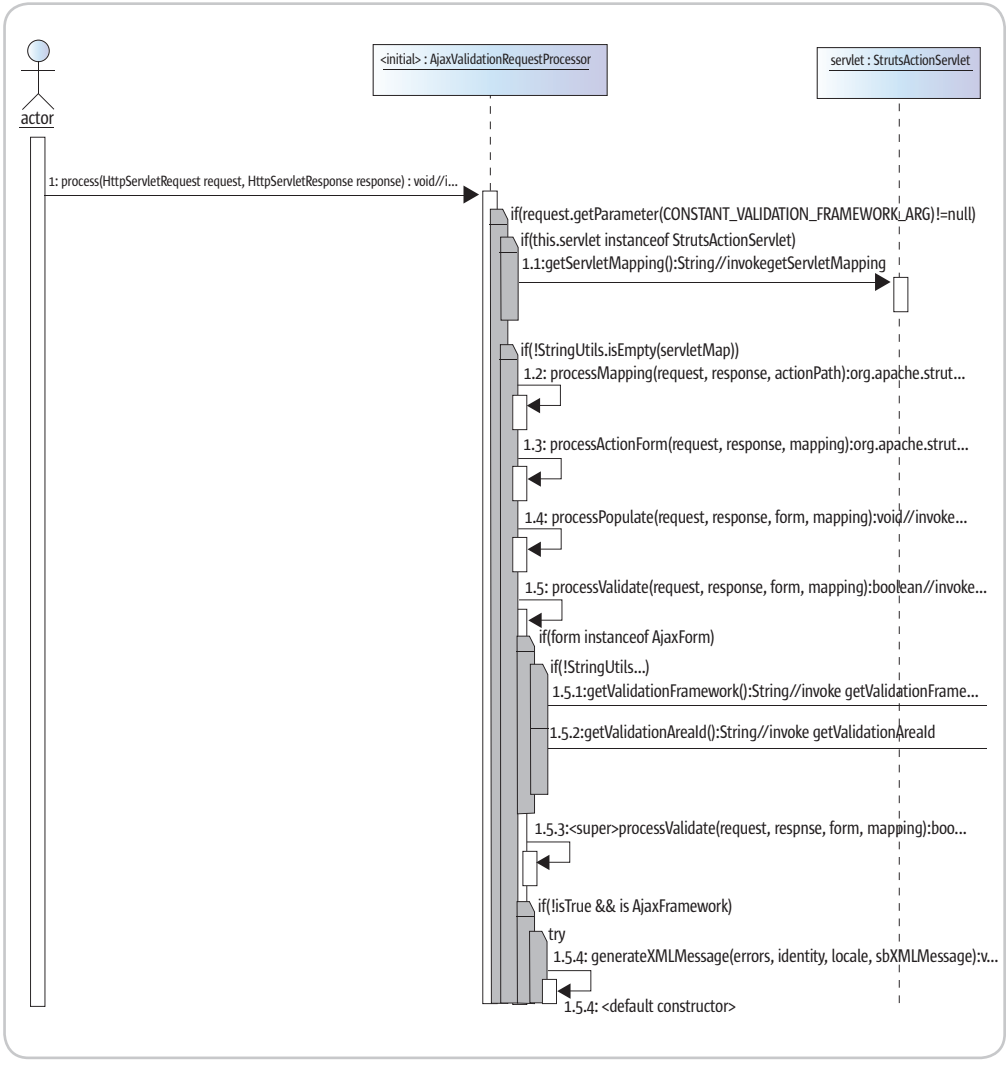


Figure 3 Controller Processing flow

> "Real-time data validation is one of the advantages of AJAX technology"

dation will be backed, the validation process should check the indicator of the validation framework being used (see Figure 3).

Generate the XML messages using JDOM as the processing engine. As shown in Figure 4, when the process validation is invoked and the condition of the validation framework is equal with the AJAX validation framework, the process will continue to populate the error message and build the XML message validation.

ErrorMessageHandler

This class handles the functionality of the XML message builder. This Java class will be building the XML message based on the identity and description property. After the caller invokes the *buildXMLMessage,* it will prepare the document and set the root element of the XML message. This class also has an *addNextXMLMessage* function to add more validation messages into the XML (see Listing 1).

The **process** method will set the content type of the response as *"text/xml"* and send the XML message as a string. The function of the process on the AjaxValidationRequestProcessor code will look like Listing 2.

The **processValidation** method will populate the action errors and construct the message based on the format contract of the XML for the client. The function of processValidation on AjaxValidationRequestProcessor code will be look like:

```
…
                ActionErrors errors =
(ActionErrors) request

.getAttribute(Globals.ERROR_KEY);
                Locale locale = (Locale)
request

.getAttribute(Globals.LOCALE_KEY);
generateXMLMessage(errors, identity,
locale, sbXMLMessage);
…
```

Client-Side Scenario

Build the Taglib Component

- ***AjaxJavaScriptLibraryTag***: The taglib component to render the JavaScript function at the client side for basic XMLHTTP controller functions.
- ***AjaxErrorHtmlRenderTag:*** The taglib component to render the area of the error message at the JSP page.

Configuring the Taglib Definition

After developing the taglib component, we need to configure the taglib tld file as shown in Listing 3.

Build JSP and Struts Configuration

To simulate the result of the validation processing, we first need to build the presentation layer by incorporating the taglib that we've build. In this case I'm trying to give an example validation by using the validation rules component from Struts, and also the validation that comes from the form itself. Prepare five textboxs under the JSP page. The first to fourth textbox are using the validation rule configuration, and the fifth textbox is using the validate process from the action form. Other than that, we also need a submit button to simulate that after submitting the form, the existing struts validation are still working without AJAX. The user interface will look like Figure 5.

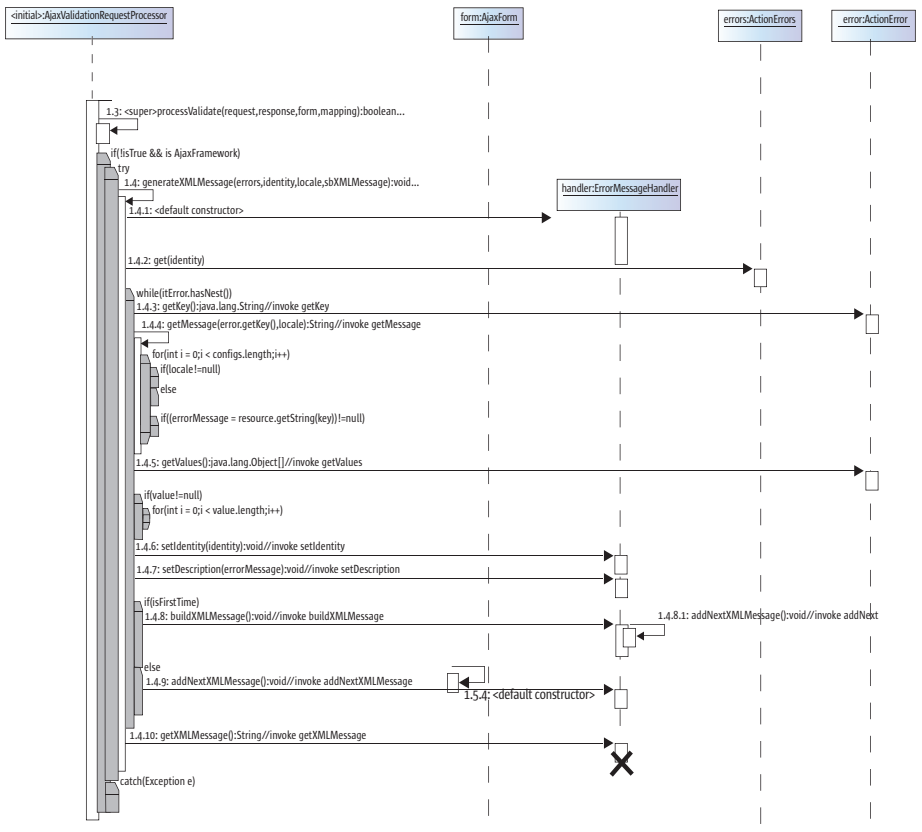
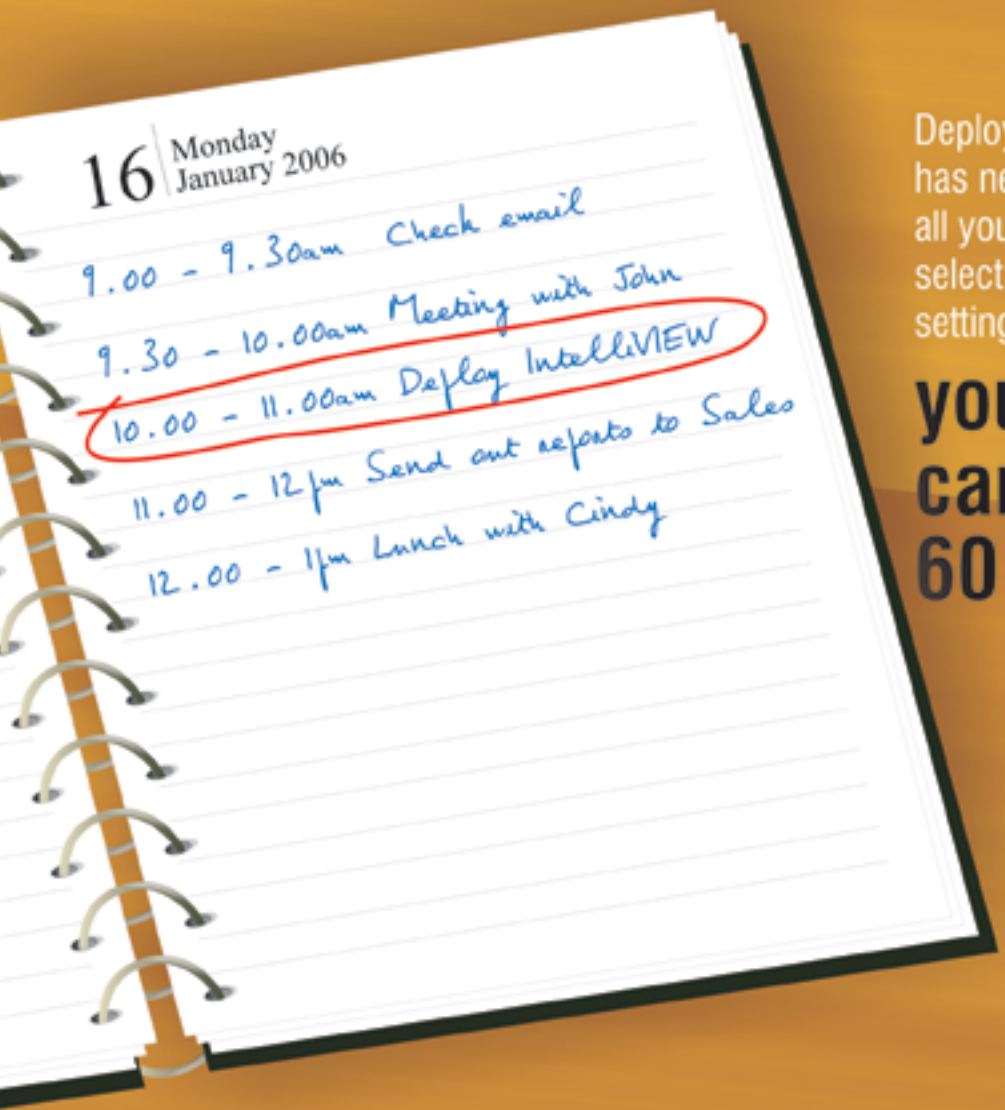


Figure 4 Filtering and XML Validation Generation Process



Deploying an enterprise reporting server has never been this easy. With IntelliVIEW, all you have to do is - launch the installer, select your database, enter the connection settings and voilà,

your first report can be out within 60 minutes!

Easy to use and even easier to deploy on servers. So you save time, effort and resources, thus providing you with a low TCO. Now that's what we mean by 'Reporting Made Easy'!

Signup NOW for the IntelliVIEW challenge. We'll install an evaluation copy on your server and create your first report, within just 60 minutes!

The IntelliVIEW Advantage

- Easy installation & integration on Java platform
- Connects to multiple RDBMS
- Quick report creation, with charts
- Range of report outputs - PDF, Excel, HTML...
- Scheduling and e-mailing
- Highly configurable 'Role Based Access Control' system
- Flexible licensing that encourages scalability
- 24x7 support
- Low TCO

Deploy IntelliVIEW... and you can be the **IT Super Hero**

Signup at **http://www.intelliview.com/jdj**
Email: sales@intelliview.com
Call toll-free **1-866-99IVIEW**

product of Synaptris

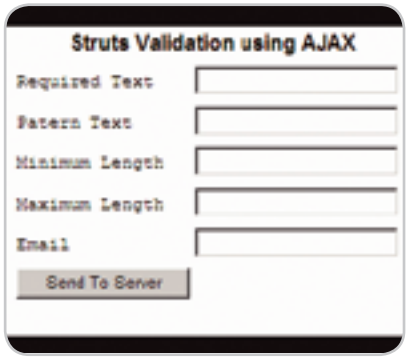
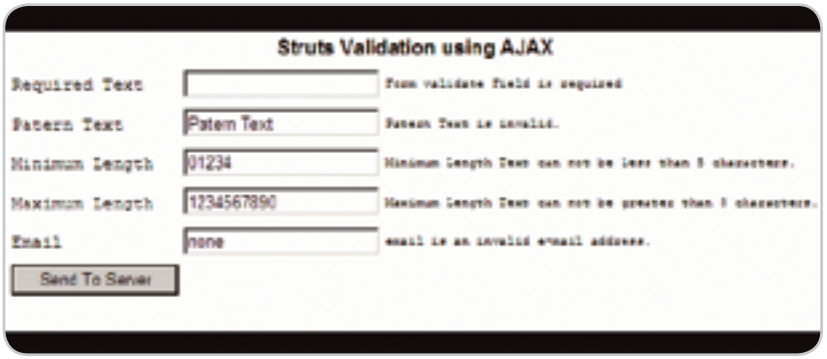


Figure 5 JSP Validation Page



Application validation test result case using AJAX inside the struts validation framework

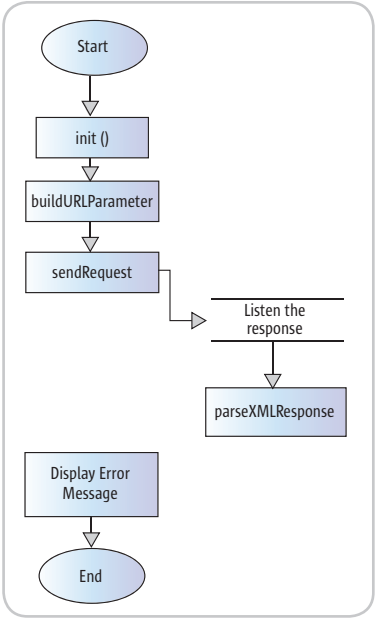


Figure 6 Client Side Processing Flow

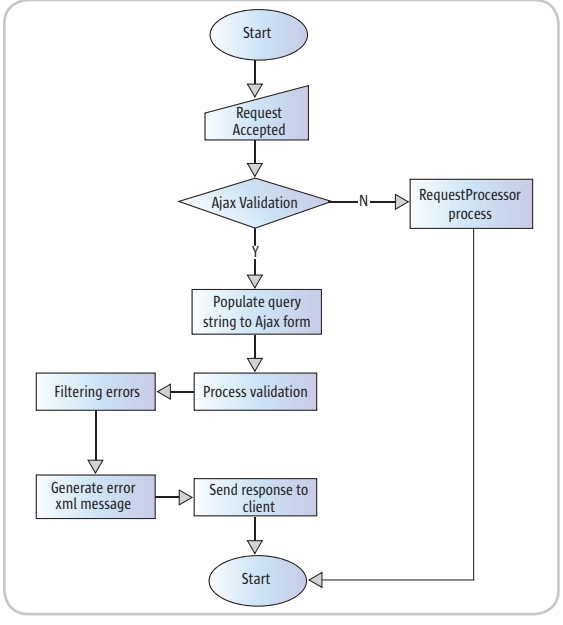


Figure 7 Server Side Processing Flow

Build Action and Action Form

For the Struts action, we just forward to the JSP we already built. The Action code will look as follows:

```
public ActionForward execute(…) {
return mapping.findForward("success");
}
```

The Action Form code will validate the requiredText property if the input is blank. Remember to extend this form from the AjaxForm class. The validate method of action form will appear as follows:

```
public ActionErrors validate(…) {
        ActionErrors errors = new
ActionErrors();
        if (StringUtils.isEmpty(this.
requiredText)) {
 errors.add("requiredText", new
ActionError("error.required.input"));

        }
        request.setAttribute(Globals.ERROR_
KEY, errors);
}
```

Applying Struts Validation Rules

The configure Struts validation rule, such as minimum length, maximum length, e-mail, and pattern text, will be applied to the input object of the client and the configuration will be similar to Listing 4.

Validation Processing Flow

First the client will initiate the XMLHTTP component to perform the request to the server and, then, on the event the user trigger starts to build, the parameter of the URL will be sent to the server. After its finish building the parameter, the client will attach the event of the oneadystatechange XMLHTTP to listen to the response from the server-side. When the response is accepted, the client side will start to parse the XML validation message and set the message into the right area (see Figure 6).

Case struts validation framework

When the request is accepted to the server, the server will start to check the parameter of the AJAX validation condition and process the validation. Once it's finished, the errors object generated will be filtered into the specific error that is related to the user input object. From this point, the XML message will be generated after the filtering process is done and sent back to the client (see Figure 7).

Summary

In this article we built a controller that has the ability to receive asynchronous requests from the client and incorporate with the struts validation process to produce the action error object. Filtering the specific input object being validated will be done after the error object produces and generates the XML message as a reply to the client-side to indicate the error message.

Altova® XMLSpy® 2006 – The industry standard XML development environment.

ALTOVA® xmlspy® 2006

Bring your development plans to light

Sneak a peek at XMLSpy® 2006, and see how essential it is to master XML.

New in XMLSpy 2006:

- Schema-aware XSLT 2.0 support
- Schema-aware XQuery support
- Updated platform integration for Microsoft® Visual Studio® .NET 2005
- Updated platform integration for Eclipse 3.1

Altova® XMLSpy, the industry standard XML development environment, is indispensable for modeling, editing, debugging and transforming all XML-related technologies. Illuminate your strategy with advanced standards compliance, extended platform integration, and enlightened usability aides. Use XMLSpy to structure XML Schemas and devise XML documents, then automatically generate runtime code from schemas in multiple programming languages. Become a markup mastermind!

Download XMLSpy® 2006 today: www.altova.com

Supports Nov. '05 W3C Candidates for XSLT 2.0, XPath 2.0 & XQuery!

ALTOVA®

Also available in the Altova XML Suite.

JAAS in the Enterprise

An integration proposal

by Raymond K. Ng
and Ganesh Kirti

Since 2001 when Java Authentication and Authorization Service (JAAS) was formally included in the Java 2 Platform Enterprise Edition (J2EE) 1.3 platform specification, the J2EE community has been grappling with the issue of JAAS/J2EE integration. On the surface, JAAS seems to be an excellent complement to J2EE: JAAS defines a pluggable Application Programming Interface (API) for authentication modules and a fine-grained Subject-based authorization model, which are both lacking in the existing J2EE security model. Since JAAS is officially part of the J2EE platform specification, it's not unreasonable to expect that you can now leverage the JAAS framework to build portable enterprise applications that have advanced authentication and authorization requirements. Unfortunately, any Java architects or developers who go down this path for their applications will soon be confronted with the harsh reality: Instead of finding a landscape defined by an unified integration architecture, they'll discover a landscape littered with incompatible vendor-specific APIs and frameworks.

If JAAS and J2EE seem to fit well together, why is there such a fragmented landscape? What has contributed to this fragmentation, and more importantly, what lies ahead? In this article we'll explore the main security issues that architects and developers must consider when designing J2EE-based solutions for enterprise deployments. We'll also highlight some of the ongoing standards-based efforts that will make it easier to include advanced authentication and authorization capabilities in your enterprise applications. Finally, we'll offer some pragmatic strategies you can consider today, while waiting for the standards to catch up.

Raymond K. Ng is a development lead at Oracle and has 11 years of industry experience. Ng currently leads the design effort of Java Platform Security in the Oracle Fusion Middleware Group at Oracle. He also serves as Oracle's Expert Group representative in JSR 115 (JACC) & JSR 196.

raymond.k.ng@oracle.com

JAAS and J2SE

JAAS is designed to bring Subject-based authentication and authorization to the Java 2 platform. On the authentication side, JAAS is modeled after the Pluggable Authentication Module (PAM)–a popular API for defining authentication modules in a portable and extensible manner. On the authorization side, JAAS inherits the rich, fine-grained, and extensible security model defined in Java 2. When JAAS was first introduced as an optional package for JDK 1.3, there was, understandably, quite a bit of excitement: "Finally, a *real* security standard for the Java platform!" (Not that code-based security isn't important, but a lot of us were waiting for a user-based authentication and authorization model.)

Indeed, JAAS 1.0 did achieve a few significant wins for Java developers. It offered a standard way to develop authentication modules (via JAAS LoginModules – see Figure 1) and a flexible way for the authentication modules to gather information dynamically from the application (via CallbackHandler – see Figure 2). It also offered a way to integrate Subject-based authorization into the Java 2 platform without affecting the code-based security model.

Of course, being hard-nosed, pragmatic people, we were also realistic – or so we thought – in our expectations: "It's only Version 1.0 of this API, and it's released as an *optional* extension, to boot. It'll probably be a few years before it goes mainstream."

Not surprisingly, when JAAS 1.0 was introduced, there were a few issues. As an optional extension, the JAAS jar files weren't generally available as part of the regular JDK distribution. In addition, its *add-on* nature was reflected in the API design – instead of having one policy that dealt with both Subject- and code-based policies, we had to deal with two separate policies (each with its own API) and some obscure *DomainCombiner* logic that updates the protection domains based on these policies at runtime.

Fortunately for Java developers everywhere, these issues largely went away when JAAS was fully integrated into the core Java security architecture in JDK 1.4: JAAS was distributed as part of the regular JDK distribution, and the Java 2 security model is now the JAAS security model – one security model, one policy API. (Please refer to Java 2 Security Architecture (http://java.sun.com/j2se/1.4.2/docs/guide/security/spec/security-spec.doc.html) for more details about JAAS/J2SE security model.)

JAAS and J2EE

Most Java developers would agree that if there were a Java development platform that could really benefit from a user-based authentication and authorization standard, it would be the J2EE platform, where security is paramount and user-based security, in particular, is essential to any enterprise-ready applications. Thus J2EE, unlike J2SE (Java 2 Platform Standard

> "It's almost impossible to write one JAAS login module that works equally well in all J2EE 1.3/1.4 containers"

SUPERCHARGE YOUR APPS WITH THE POWER OF LOCATION INTELLIGENCE

100% Java SDK enables Location Intelligence through web services

Create desktop & web UIs using Swing, JSP and JSF Components

Integration with Eclipse, NetBeans, IntelliJ and more

Create applications for:

- Web-based store location finders
- Analyzing where revenue comes from – and where it doesn't
- Visualizing where your customers are
- Managing assets such as cell towers, vehicles and ATMs

Try it and see for yourself. **Visit www.mapinfo.com/sdk**
Learn more about the MapInfo Location Platform for Java, access whitepapers and download free SDKs.

MapInfo
Be Location Intelligent™

Contact us at sales@mapinfo.com

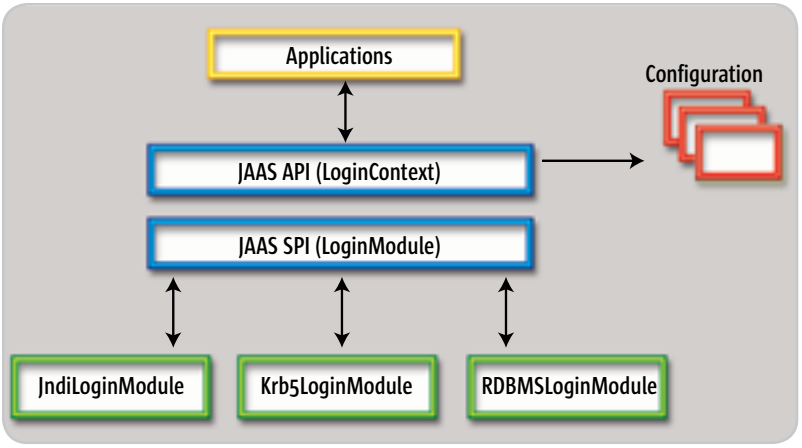


Figure 1 JAAS authentication architecture

Edition), came with a rather sophisticated user-based security model since J2EE 1.0. With the introduction and subsequent integration of JAAS into the J2SE platform, the main question has now become: "How do we integrate JAAS into J2EE?" (This is a question Graham Hamilton posed to the Java security experts at a one-day Java security summit held in Boston shortly after JAAS 1.0 was introduced.)

On the surface, it appears that JAAS and J2EE complement each other very well. Although J2EE defines a declarative way to configure the authentication method to be used with a Web/EJB module, it doesn't define a pluggable authentication module API; JAAS (with its LoginModule API) nicely fills this void. And although J2EE authorization serves quite well the security needs of typical (security-unaware) Web-based and EJB-based applications, it's also a rather coarse-grained and inflexible model (the only privileges supported are URL-based and EJB method-based). The JAAS/Java2 permission-based model is extensible and can model arbitrarily complex and fine-grained authorization policies.

Given these considerations, it wasn't surprising that shortly after its introduction, JAAS 1.0 was included as part of the J2EE 1.3 platform specification. Unfortunately — perhaps due to time and resource constraints — there were no provisions in the J2EE 1.3 specification regarding *how* JAAS 1.0 was to be integrated into J2EE-compliant containers. In other words, it'd be up to each container vendor to figure out the right integration strategy. Inevitably, because different vendors decided to do this in different enough ways, the fact that JAAS 1.0 was officially part of J2EE 1.3 isn't very meaningful to Java developers — at least to those who care about developing *portable* J2EE applications.

- ***Consider authentication:*** While it's possible to write a JAAS-based login module that works properly in a J2EE 1.3/1.4-compliant container, it'd be difficult – in fact, almost impossible – to write one JAAS login module that works equally well in all J2EE 1.3/1.4 containers.
- ***Consider authorization:*** While it would be possible for a J2EE application to use the JAAS policy API and Java 2 permission model to handle fine-grained authorization policies, it'd be impossible for such an application to retrieve the JAAS Subject (representing the currently authenticated user) in a portable manner.

JAAS/J2EE Integrated: Secure Enterprise Applications

Before we get into too much detail, let's step back and ask ourselves: "What would a truly JAAS/J2EE integrated architecture look like?" Put it this way. If you were the project manager responsible for this integration project, what would be on your project's wish list?

First, let's consider authentication: JAAS has defined a portable, extensible API for authentication modules that lets you reuse a well-behaved LoginModule implementation in different contexts. What does this mean in a J2EE context? We believe a reasonable goal for a JAAS/J2EE integration architecture would be to let any JAAS-compliant login module be plugged into any J2EE-compliant container — *without any modifications*. In other words, you should be able to develop or buy/download an RDBMSLoginModule to use with your applications and expect it to work well in all J2EE-compliant application servers that might end up hosting your applications. Given certain reasonable constraints (for example, the RDBMSLoginModule shouldn't use any protocol-specific Callbacks), it should also work equally well for most of the managed resources (such as servlet/JSP, EJB, or MBeans) over a variety of connection protocols (such as HTTP, RMI/IIOP, or JMX/RMI).

Now, let's consider authorization. We believe a properly designed integration architecture should cover the following: First, the J2EE authorization model should work properly in a JAAS/J2EE context. In other words, using a JAAS LoginModule to handle authentication for your application doesn't mean you want to get rid of the nice declarative security constraints you've worked hard to *exactly* fit your application. Also, the managed component (such as Servlet/JPS, EJB, or MBeans) should be invoked in an execution context where the currently authenticated user (represented by a JAAS Subject instance) is readily available (via JAAS API) to the component. (In standard JAAS, this typically means the managed component is invoked within a Subject.doAs[Privileged] block.) Finally, the legacy J2EE authorization API (such as isCallerInRole/getCallerPrincipal) should work properly in a JAAS/J2EE context; this means that if your application is security-aware and uses isCallerInRole to do advanced role-based processing, your application should continue to work in a JAAS/J2EE-integrated environment.

So far, we've only outlined the basic requirements; these ensure that JAAS and J2EE play well with each other, and that the legacy applications or modules developed with either context in mind continue to work well in an integrated environment. A JAAS/J2EE integration framework that

Ganesh Kirti is a senior software development manager at Oracle with 11 years of industry experience. He currently leads development of Java Platform Security in the Oracle Fusion Middleware Group at Oracle. Ganesh has a wide range of engineering experience including developing identity management and SOA security products.

ganesh.kirti@oracle.com

implements these requirements in a reasonable manner might be considered a solid investment, but probably not something worth jumping up and down about.

Yet these basic requirements do bring about a level of power and sophistication not seen before in J2EE security. For instance, by leveraging the integrated JAAS/J2EE architecture, you can finally build advanced enterprise applications that authenticate via a custom (application-specific) user repository (via standard JAAS LoginModules); leverage a coarse-grained J2EE authorization model for pre-dispatch authorization decisions; and leverage a customized, fine-grained JAAS permission model to protect application-specific resources. And you achieve all of these *without sacrificing portability*.

This is nothing to be sneezed at – and if you're like us, you're probably salivating at the possibilities this combination has opened up for your application architecture...

Not so fast. Yup — there's a catch.

The Current Landscape

Equipped with this modest list of requirements, we're ready to evaluate the current landscape. As an industry, how do we fare in this context? In a nutshell, not very good. In fact, if we use this basic requirement list as a scorecard, most vendors wouldn't do well at all.

One thing worth noting about these basic requirements. Modest as they are, *none of them* are addressed in J2EE 1.3! No wonder the current JAAS/J2EE integration landscape isn't pretty. (Unfortunately, even with JACC (JSR 115) being part of J2EE 1.4, many of these requirements are still not met. We'll discuss the role of JACC later in this document.) We can and *should* do better.

We don't want to speculate why and how this came to be, but we can assume there *were* good reasons to incorporate JAAS 1.0 *as is* into J2EE 1.3 without spelling out all the integration details. By taking this approach, as vendors gain experience by doing the real integration work, the experience will be taken into consideration in the next J2EE update. In fact, we think that's what happened with the JAAS/J2EE integration process. The real integration work has only just begun.

Standardization Efforts

Slowly but surely, the standardization process is beginning to fill some of the voids left out in J2EE 1.3. We'll examine the two main JSRs (Java Specification Requests) directly related to the topic under discussion. Not surprisingly, there are primarily two relevant JSRs. One deals with authentication and the other with authorization.

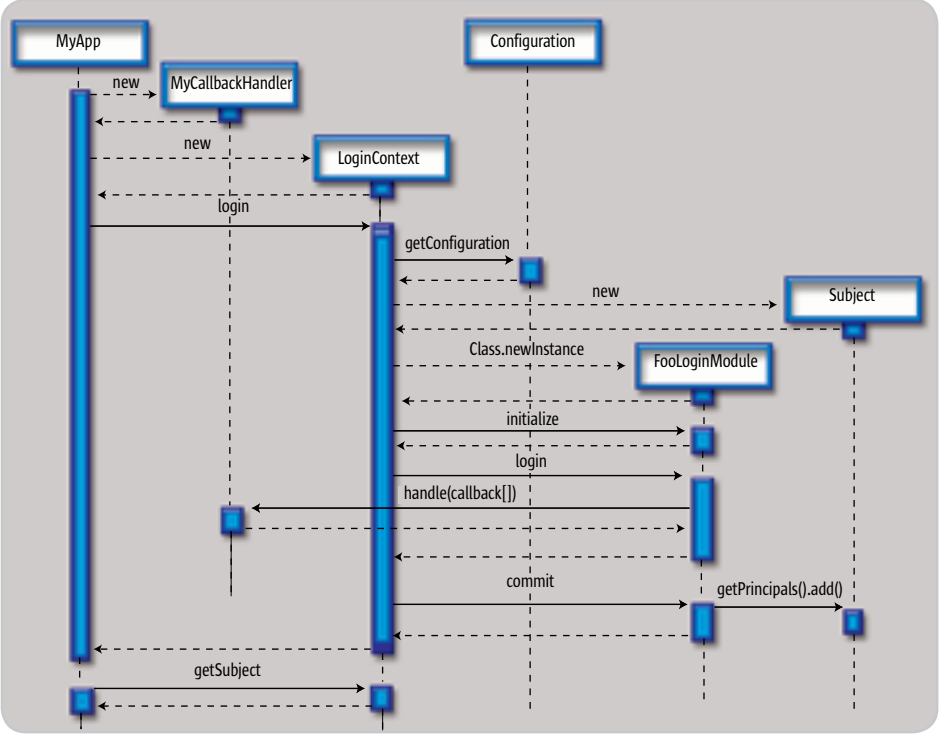


Figure 2 Sample JAAS authentication call sequence

JSR 115 Java Authorization Contract for Containers (JACC)

Incorporated as part of the J2EE 1.4 specification, the explicit intent of JSR 115 is as follows:

Define new java.security.Permission classes to satisfy the J2EE role-based authorization model. The specification will define the binding of container access decisions to operations on instances of these permission classes. The specification will define the semantics of policy providers that employ the new permission classes to address the authorization requirements of J2EE.

In a nutshell, this specification aims to consolidate the J2EE and JAAS authorization models by defining Java 2 permission classes that capture J2EE security semantics.

Since this specification is adopted as part of J2EE 1.4, any J2EE 1.4-compliant containers have built-in JACC support.

As a first release, JACC has achieved its primary goals — i.e., to define Permission classes for J2EE security constraints (thereby consolidating the authorization models) and to define a standard contract between the container and the policy provider (thereby achieving limited interoperability). On the other hand, JACC 1.0 — just like any 1.0 specifications — isn't without issues.

JACC 1.0 doesn't clarify how J2EE API such as getCallerPrincipal (which assumes a single java.security.Principal instance to be returned) should behave when multiple Principals are associated with the Subject representing the currently authenticated user as a result of JAAS authentication.

By affording the container a degree of flexibility regarding how authorization decisions are to be done, it's not clear how an application can reliably retrieve the Subject representing the currently authenticated user in a portable manner.

> A properly defined JAAS/J2EE integration architecture provides a flexible and powerful foundation on top of which sophisticated, *secure and portable* enterprise-level applications are made possible"

The authorization SPI as defined in JACC 1.0 is somewhat incomplete, thus compromising interoperability – specifically, it doesn't define a standard way to map J2EE logical roles to deployment principals. The definition of a standard role-to-principal mapping facility will be critical to achieve true PnP (plug-and-play) for JACC policy providers.

While there are still issues to be hashed out, we view JSR 115 as a step in the right direction. We expect future updates of JSR 115 that address the issues we've identified here.

JSR 19: Java Authentication Service Provider Interface for Containers

The explicit goal of this JSR is as follows:

The proposed specification will define a standard service provider interface by which authentication mechanism providers can be integrated with containers. Providers integrated through this interface will be used to establish the authentication identities used in container access decisions, including those used by the container to invock components in other containers. The specification will define standard interfaces between containers and authentication modules...

Thus JSR 196 can be viewed as the authentication equivalent of JSR 115. It seeks to define a standard SPI via which authentication modules can be integrated with containers – much like how JSR 115 defines a standard authorization SPI through which policy modules are integrated with containers.

As we write this article, only an early draft is available for review. Contained in this draft, however, is a chapter dedicated to JAAS integration called "LoginModule Bridge Profile." In this chapter a standard approach to integrate JAAS LoginModules with JSR 196 authentication modules is articulated. In so doing, this specification has the potential to effectively fill another void we have identified earlier – the need to standardize how a JAAS LoginModule is integrated with containers.

Note that while this JSR is targeted at J2EE 1.4 and above, the JSR is currently lagging behind the original proposed schedule and is not part of J2EE 5 (nor does its inclusion appear likely, as J2EE 5 is already in "proposed final draft" stage).

Nonetheless, the eventual adoption of JSR 196 should be good news to those who have invested in JAAS architecture for authentication. We look forward to working with the specification leads to ensure that the JAAS LoginModule integration is properly defined in the specification.

JAAS/J2EE Integration Strategies for Today

So you're an enterprise application architect and you see the clear benefits of an integrated J2EE/JAAS integration architecture. You know the standards are heading in the right direction, but you need a solution that works today. What are your options?

Obviously, you can stick to one vendor's implementation and migrate to standard-based containers when the standards catch up with your needs. The upside of this approach is that there's little bootstrapping cost and you can start development right away. The downside is that your application is locked-in to a specific vendor's API — and you'll ultimately pay the price of either sticking with your container vendor no matter what or re-writing your application when the standards are ready. Obviously, the extent to which your vendor of choice fulfills the requirements identified in this article, the better insulated your application will be from vendor-specific lock-in.

Another approach is to add an additional level of indirection. Instead of sticking with one vendor's API, you devise a thin wrapper layer for a small number of APIs (such as the equivalent of Subject.getSubject), which insulates your application from vendor-specific APIs. The upfront cost of this approach is obviously a bit higher — you'll have to design the wrapper layer, after all — but the benefit is also obvious. Your application will be shielded from vendor lock-ins and it'd be relatively easy to deploy it to a different vendor. (This assumes that the thin wrapper layer is properly designed so that it's configurable and it defines a SPI (Service Provider Interface) layer that lets you add plug-ins for different container vendors.) In general, we recommend this approach if you can afford the upfront cost.

Finally, you can search the Internet for open source or commercial projects that take care of this for you. At the time of this writing, there weren't many portable JAAS/J2EE integration frameworks (open source or otherwise) available, but things change, so keep an eye out for new efforts in this direction.

Conclusion

JAAS and J2EE are complimentary technologies that play well together. We feel that a properly defined JAAS/J2EE integration architecture provides a flexible and powerful foundation on top of which sophisticated, *secure and portable* enterprise-level applications are made possible. Though today's landscape is somewhat less than perfect, the industry experts are hard at work to resolve the issues identified in this article. It's our hope that a reasonably complete JAAS/J2EE architecture will emerge out of the standard bodies in the near future.

For the Java architects and designers that need a solution *today*, fear not – you don't need to sit idly waiting for the standards to catch up. With some careful forethought and strategic thinking, we believe it's possible to design secure JAAS/J2EE-based applications that work in today's containers – and perhaps more importantly – in tomorrow's containers as well.

Introduction to Acegi

Mastering the security framework

by David Hardwick

I recently evaluated the use of Acegi as the security framework for a Web development project. In the end, we decided to move forward with Acegi but in the beginning it took a couple days to come to that decision. The amazing thing is: once you get over the initial learning curve, it's smooth sailing. Hence, I wanted to share my experiences with it because first, I wanted to expose the Acegi security framework to *JDJ* readers and, second, I wanted to make it easier for *JDJ* readers to get over the initial learning curve. Once you're over that, you should be well on your way with Acegi.

Exposing Acegi Security Framework

Acegi is an open source security framework that allows you to keep your business logic free from security code. Acegi provides three main types of security:

1. Web-based access control lists (ACL) based on URL schemes
2. Java class and method security using AOP
3. Yale's Central Authentication Service for single sign-on (SSO).

Acegi also provides the option of performing container security.

Acegi uses Spring as its configuration settings, so those familiar with Spring will be at ease with Acegi configuration. If you're not familiar with Spring, it's still easy to learn Acegi configuration by example. You don't have to use SpringMVC to secure your Web application. I have successfully used Acegi with Struts. You can use Acegi with WebWork and Velocity, Struts, SpringMVC, JSF, Web Services, and more.

Why use Acegi instead of JAAS? It can be difficult to stray from well-documented standards like JAAS. However, porting container-managed security realms is not easy. With Acegi, this security layer is an application framework that is easily ported. Acegi will allow you to easily reuse and port your "Remember Me," "I forgot my password," and log-in/log-out security functions to different servlet and EJB containers. If you have a standards-based security layer that you have re-created for numerous Java applications and it is *not* getting reused, you need to take a good look at Acegi. Besides, why are you spending time on framework coding when you should be focusing on the business logic? Leave the framework development to product developers and the open source community.

David Hardwick is a technology manager at Sapient Corporation, a business innovator. He has nearly 10 years of application development experience in commercial, government, and non-profit sector industries.

dhardwick@sapient.com

Getting Over That Initial Learning Curve

To get you over the initial learning curve, I'll take you through a simple setup using a demonstration application. I'll focus on the first security approach – URL-based security for Web applications because that's the most commonly used.

Installation

First things first – we need to install it! I'll use Tomcat 5 as my servlet container to illustrate.

Step 1: Set up a new Tomcat *Web context* with the "WEB-INF/", "WEB-INF/lib/", and "WEB-INF/classes" folders per usual (see Figure 1). I called my context "/acegi-demo" and access it using http://localhost:8080/acegi-demo/.

Step 2: Add another folder called "/secured," which we'll protect with Acegi.

Step 3: Now let's add the necessary Acegi library files to plug-in Acegi to our Tomcat context. (Please download the acegi-demo.zip file provided with this article from http://jdj.sys-con.com.)

Let's understand the JAR packages we are adding to the *lib* directory. The most important JAR is *acegi-security-0.8.3.jar,* the Acegi core library. Acegi leverages Spring for its configuration, so we also need *spring-1.2.RC2.jar.* The remaining JARs are utilities libraries for dealing with collections (*commons-collections-3.1.jar*), logging (*commons-logging-1.0.4.jar, log4j-1.2.9.jar*), and regular expressions (*oro-2.0.8.jar*). Special thanks to Apache Jakarta for these wonderful utility libraries.

Configuration

Now that we have our core infrastructure in place, let's focus on configuration.

Step 4: Configure the web.xml file (see Listing 1) to begin tying the Web application to the Acegi security framework.

1. First, we need to set up two parameters: *contextConfiguration*, which will point to Acegi's configuration file, and *log4jConfigLocation*, which will point to Log4J's configuration file.
2. Next, we have to set up the *Acegi Filter Chain Proxy*; this critical proxy allows Acegi to interact with the servlet filtering feature. We will talk about this more in step 5 (configuring *applicationContext.xml*).
3. Finally, we want to add three listeners to loosely couple Spring with the Web context, Spring with Log4J and Acegi with the HTTP Session events in the Web context, such as *create session* and *destroy session*.

Step 5: Now we need to configure the *applicationContext.xml* (see Listing 2) to instruct the Acegi framework to perform our security requirements. It is important to note that you typically don't have to write or compile any code to fuse your application with the Acegi security framework. Acegi is almost entirely configuration driven, thanks to a great design by its creator, Ben Alex, and Spring. Okay, enough back patting, let's get to it…

Remember, the Acegi Filter Chain Proxy is critical. This is the backbone of the configuration. Using the servlet filter

specification, Acegi is able to plug in its security functionality in a modular way.

I ordered the Spring bean references in the *applicationContext.xml* file based on the sequence each bean is referenced, starting with the *filterChainProxy* bean. If you are new to Spring, just know that the order in which a bean is referenced is not important. I ordered it this way to make it as easy as possible to follow along.

```
<bean id="filterChainProxy" class="net.
sf.acegisecurity.util.FilterChainProxy">
      <property name="filterInvocationDefi
nitionSource">
          <value>
     CONVERT_URL_TO_LOWERCASE_BEFORE_
COMPARISON
     PATTERN_TYPE_APACHE_ANT
     /**=httpSessionContextIntegrationFil-
ter, authenticationProcessingFilter, anony-
mousProcessingFilter, securityEnforcement-
Filter
          </value>
      </property>
    </bean>
```

In the *filterChainProxy* bean (see code snippet above), we tell Acegi that we want to use lowercase for all URL comparisons and use the Apache ANT style for pattern matching on the URLs. In our example, we run the *filterChainProxy* on every single URL by specifying */**=Filter1,Filter2, etc.* Next, we set up the filter chain itself, *where order is very important*. We have four filter chains in this simple example, but when you start using Acegi, you'll most likely have more. Viewing *applicationContext.xml*, please take a few moments to follow all the bean references in great detail as you traverse the filter chain. I will walk through each item in the filter chain at a high level.

The first item in the chain must be the *httpSessionContextIntegrationFilter* filter. This filter works hand-and-hand with the HTTP Session object and the Web context to see if the user is authenticated and, if so, then what roles the user has. We have little to configure for this filter.

The second item in the chain is the *authenticationProcessingFilter* filter, which searches for any URL that matches */j_acegi_security_check* because this is the URL that our login form will post a username and password to when attempting authentication. This filter also contains the configuration information detailing where to send someone if the login succeeds or fails. If it succeeds, you can configure this filter to direct the user to the page the user originally tried to access or direct the user to a particular start page where you want all authenticated users to land after authentication. I have the latter option configured in my example by setting *alwaysUseDefaultTargetUrl* to true and you just set it to false to get the former option.

One of the beans configured in the *authenticationProcessingFilter* is the *authenticationManager* bean. This bean manages the various providers you configure. A *provider* is essentially a repository of usernames with corresponding passwords and roles. The *authenticationManager* will stop iterating through the list of providers once a user is successfully authenticated. In practice, you may have two or three providers; for example, one provider could access an Active Directory for employee credentials, while your second provider might access a database for customer credentials. You will most often need an *anonymousAuthenticationProvider* because you need it to allow access to pages that do not requiring authentication to access, such as the login page or the home page. The demonstration application for this article uses a *memory* provider and an *anonymous* provider. Once you get this simple application working, you probably want to add a JDBC or LDAP provider.

The third item in the chain is the *anonymousProcessingFilter* filter. This will match the value created by the *anonymousAuthenticationProvider*.

The fourth and final item in the filter chain is the *securityEnforcementFilter* filter. This filter has two beans: the *filterSecurityInterceptor* and the *authenticationProcessingFilterEntryPoint*. The latter bean is used to direct the user to the login form each time the user tries to access a secured page but is not logged in. We can also force the user to use HTTPS. The former bean, *filterSecurityInterceptor*, does quite a bit of heavy lifting by tying all our filters together.

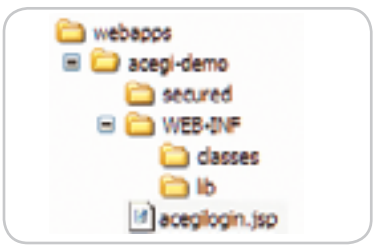


Figure 1 Folder Structure

The *filterSecurityInterceptor* bean checks that the authenticated user has the right roles (or permissions) to access a particular *objectDefinitionSource*. Here we are using *AffirmativeBased* voting, which means the user just has to have one of the roles specified in the *objectDefinitionSource*. This is most likely what you will use, but Acegi does have a unanimous voter that ensures that a person has every role specified in the *objectDefinitionSource* before granting access. By now you may have realized that *objectDefinitionSource* determines who can access what.

The *objectDefinitionSource* starts off with the same two configuration instructions that *filterChainProxy* did, namely converting all URLs to lowercase and using the Apache ANT style for regular expressions. Next, we define which roles are allowed to access a particular URL. In our example, we give anonymous access to the */acegilogin.jsp* page so that unauthenticated users can arrive at this page to log in. The next line in the *objectDefinitionSource* provides access to everything below the */secured* directory for any user with the ADMIN role. Finally, we add a line that starts with /** to match on every URL. The filter will stop once the URL matches on a URL, so make sure you put specific regular expressions toward the top and broad regular expressions toward the bottom to ensure you get the desired behavior. If you were working with Struts, you could either set up your struts in modules http://struts.apache.org/struts-core/userGuide/configuration.html#5_4_1_Configure_the_ActionServlet_Instance or simply specify the StrutAction (e.g., /CustomerAdd.do) in the *objectDefinitionSource*.

At this point, we are done with applicationContext.xml file. To complete our demonstration application, all we need to do now is create a login form and put something in the */secured* directory to see that our Acegi authentication and authorization configuration is working. (See the acegi-demo.zip for */acegilogin.jsp* and */secured/index.jsp*.)

The login form is very simple; it has input fields for the username and password, *j_username* and *j_password*, respectively, and a form action pointing to *j_acegi_security_check* since that is what the *authenticationProcessingFilter* filter listens for to capture every login form submission.

Test your configuration and inspect the Tomcat logs and the Log4J log file that we configured for this application if you run into problems.

Now That I'm Over the Initial Learning Curve, What's Next?

Once you have this simple Acegi demonstration application running, you will undoubtedly want to increase its sophistication. The first thing I would want to do is to add a JDBC profile in addition to the simple in-memory profile.

I can understand the excitement after getting the initial application up and running, but you still have some reading to do in order to eclipse the initial learning curve. Read through the articles posted in the External Web Articles section of the Acegi Web site http://acegisecurity.sourceforge.net. Read through the Reference Documentation provided by Ben Alex, the creator of Acegi. Ben does a good job of providing help through the support forum too. Also, read the well-kept JavaDocs as your main source of information once you get familiar with Acegi. Of course, you can opt to read the source code – it's open source!

Since this is your first time using Acegi, test after each change to the *applicationContext.xml file*. The process of "one change, then test" will help you understand exactly what change to the applicationContext.xml file caused an error if one should occur. If you make four changes to that file, restart the application and get an error, then you won't know which one of the four changes caused the error.

Note that I kept this application very simple. As you add in features such as Acegi's caching, you will need to add the appropriate libraries (or JARs). Look at the Acegi example application available on the Acegi Web site to get access to all the various libraries. The example application on the Acegi Web site is complex, so it is not the best place to start to get over the initial learning curve, unfortunately, hence my attempt to make it easier with the article!

No Groups in Acegi?

Acegi will let you work with the notion of groups. When you put a person in a group, you are just grouping the permissions (or roles) that the group does or does not have. So, when you set up your LDAP or JDBC profile, you need to make sure that the query returns the roles that the users' groups should have access to.

Conclusion

Acegi is a very configurable, open source security framework that will finally let you reuse and port your security layer components. It can be daunting at first, but this article should easily remove the stress in getting over the learning curve. Remember, you need to get this simple application running, test after each change, and read the recommended readings to fully surmount the initial learning curve. After you follow these steps, you will be well on your way to mastering Acegi.

I welcome all feedback and/or suggestions for further aspects of Acegi to cover in future articles.

Listing 1

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE web-app
  PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
  "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">

<web-app>
  <display-name>Acegi Demo</display-name>

<!-- 1. Setup two parameters:           -->
<!--    a) Acegi's configuration file   -->
<!--    b) Loggin configuration file    -->
   <context-param>
 <param-name>contextConfigLocation</param-name>
 <param-value>
  /WEB-INF/applicationContext.xml
 </param-value>
   </context-param>

   <context-param>
 <param-name>log4jConfigLocation</param-name>
 <param-value>/WEB-INF/classes/log4j.properties</param-value>
   </context-param>

<!-- 2. Setup the Acegi Filter Chain Proxy -->
   <filter>
        <filter-name>Acegi Filter Chain Proxy</filter-name>
        <filter-class>net.sf.acegisecurity.util.FilterToBeanProxy</
filter-class>
        <init-param>
            <param-name>targetClass</param-name>
            <param-value>net.sf.acegisecurity.util.
FilterChainProxy</param-value>
        </init-param>
   </filter>

    <filter-mapping>
      <filter-name>Acegi Filter Chain Proxy</filter-name>
      <url-pattern>/*</url-pattern>
    </filter-mapping>

<!-- 3. Setup three listeners -->
<!--    a) Setup a listener to connect spring with the web context
-->
    <listener>
 <listener-
class>org.springframework.web.context.ContextLoaderListener</lis-
tener-class>
    </listener>

<!-- b) Setup a listener to connect spring with log4J -->
    <listener>
 <listener-class>org.springframework.web.util.Log4jConfigListener</
listener-class>
    </listener>

<!--    c) Setup ACEGI to subscribe to http session events in the
web context -->
    <listener>
        <listener-class>net.sf.acegisecurity.ui.session.HttpSessionE
ventPublisher</listener-class>
    </listener>

<!-- 4.  The Usual Welcome File List -->
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>

</web-app>
```

Listing 2

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN" "http://www.spring-framework.org/dtd/spring-beans.dtd">

<beans>

   <bean id="filterChainProxy" class="net.sf.acegisecurity.util.FilterChainProxy">
      <property name="filterInvocationDefinitionSource">
         <value>
     CONVERT_URL_TO_LOWERCASE_BEFORE_COMPARISON
     PATTERN_TYPE_APACHE_ANT
     /**=httpSessionContextIntegrationFilter, authenticationProcessingFilter, anonymousProcessingFilter, securityEnforcementFilter
         </value>
      </property>
    </bean>


  <!-- The first item in the Chain: httpSessionContextIntegrationFilter -->
  <bean id="httpSessionContextIntegrationFilter" class="net.sf.acegisecurity.context.HttpSessionContextIntegrationFilter">
      <property name="context">
       <value>net.sf.acegisecurity.context.security.SecureContextImpl</value>
      </property>
  </bean>


  <!-- the second item in the chain: authenticationProcessingFilter -->
   <bean id="authenticationProcessingFilter" class="net.sf.acegisecurity.ui.webapp.AuthenticationProcessingFilter">
      <property name="authenticationManager"><ref bean="authenticationManager"/></property>
      <property name="authenticationFailureUrl"><value>/acegilogin.jsp?login_error=1</value></property>
      <property name="defaultTargetUrl"><value>/secured/</value></property>
      <property name="alwaysUseDefaultTargetUrl"><value>true</value></property>
      <property name="filterProcessesUrl"><value>/j_acegi_security_check</value></property>
   </bean>

   <bean id="authenticationManager" class="net.sf.acegisecurity.providers.ProviderManager">
      <property name="providers">
         <list>
     <ref bean="daoAuthenticationProvider"/>
     <ref local="anonymousAuthenticationProvider"/>
        </list>
      </property>
   </bean>

   <bean id="daoAuthenticationProvider" class="net.sf.acegisecurity.providers.dao.DaoAuthenticationProvider">
      <property name="authenticationDao">
         <ref local="memoryAuthenticationDao"/>
      </property>
   </bean>

   <bean id="memoryAuthenticationDao" class="net.sf.acegisecurity.providers.dao.memory.InMemoryDaoImpl">
      <property name="userMap">
        <value>sapient=password,ROLE_ADMIN,ROLE_USER</value>
      </property>
   </bean>

   <bean id="anonymousAuthenticationProvider" class="net.sf.acegisecurity.providers.anonymous.AnonymousAuthenticationProvider">
      <property name="key"><value>foobar</value></property>
   </bean>


   <!-- the third item in the chain: anonymousProcessingFilter -->
   <bean id="anonymousProcessingFilter" class="net.sf.acegisecurity.providers.anonymous.AnonymousProcessingFilter">
      <property name="key"><value>foobar</value></property>
      <property name="userAttribute"><value>anonymousUser,ROLE_ANONYMOUS</value></property>
   </bean>

   <!-- the fourth item in the chain: securityEnforcementFilter -->
   <bean id="securityEnforcementFilter" class="net.sf.acegisecurity.intercept.web.SecurityEnforcementFilter">
      <property name="filterSecurityInterceptor"><ref local="filterInvocationInterceptor"/></property>
      <property name="authenticationEntryPoint"><ref local="authenticationProcessingFilterEntryPoint"/></property>
   </bean>


   <bean id="filterInvocationInterceptor" class="net.sf.acegisecurity.intercept.web.FilterSecurityInterceptor">
      <property name="authenticationManager"><ref bean="authenticationManager"/></property>
      <property name="accessDecisionManager"><ref local="httpRequestAccessDecisionManager"/></property>
      <property name="objectDefinitionSource">
         <value>
        CONVERT_URL_TO_LOWERCASE_BEFORE_COMPARISON
        PATTERN_TYPE_APACHE_ANT
        /acegilogin.jsp*=ROLE_ANONYMOUS,ROLE_USER,ROLE_ADMIN
        /secured*=ROLE_ADMIN
        /**=ROLE_USER
         </value>
      </property>
   </bean>

   <!-- authenticationManager defined above -->

   <bean id="httpRequestAccessDecisionManager" class="net.sf.acegisecurity.vote.AffirmativeBased">
      <property name="allowIfAllAbstainDecisions"><value>false</value></property>
      <property name="decisionVoters">
         <list>
            <ref bean="roleVoter"/>
         </list>
      </property>
   </bean>

   <bean id="roleVoter" class="net.sf.acegisecurity.vote.RoleVoter"/>

  <bean id="authenticationProcessingFilterEntryPoint" class="net.sf.acegisecurity.ui.webapp.AuthenticationProcessingFilterEntryPoint">
      <property name="loginFormUrl"><value>/acegilogin.jsp</value></property>
      <property name="forceHttps"><value>false</value></property>
  </bean>

   <!-- Done with the chain -->

   <!-- This bean automatically receives AuthenticationEvent messages from DaoAuthenticationProvider -->
   <bean id="loggerListener" class="net.sf.acegisecurity.providers.dao.event.LoggerListener"/>

</beans>
```

keep your J2EE problems...at bay. Get freedom for more in life

No more endless hours of sorting out J2EE problems. AutoPilot ™ from Arcturus Technologies, automatically does that for you.

It's an affordable new concept that eliminates your worries about WebLogic upkeep and maintenance. AutoPilot ™ is so powerful and efficient that not only does it monitor, analyze and optimize your WebLogic for maximum performance, it also advises you about how to cure problems and avoid costly failures. It does this all automatically, giving you the freedom to put back what you have been missing in life.

- Automatically optimizes WebLogic server, saving you time and effort.
- Detects problems automatically.
- Gives advice from a dynamic knowledge base.
- Provides WebLogic system state for analysis in the event of a catastrophic failure.
- Optimizes existing J2EE resources.
- Increases the life, effectiveness, and quality of existing resources by reducing or eliminating problem areas.
- Improves resource communication and response time.
- Enhances server capacity to handle more loads and transactions.
- Provides advanced system monitoring.
- Furnishes customizable reports.
- Generates email alerts.
- Cost Effective at only $999 per IP address, regardless the number of CPUs

Want to know more & place an order ?
Log on to **www.autopilot2005.com**

AUTOPILOT™
AUTO OPTIMIZER

Experiences with the New 1.5 Java Language Features

Understand when and how to use them

by Jess Garms and Tim Hanson

Quite a few articles have been written introducing the new language features in JDK 1.5. In this article, we're going to go a little deeper and provide tips on how to effectively use those features.

Introduction

During the beta period for JDK 1.5, we worked on a 1.5 Java compiler for BEA's Java IDE. As we implemented various new features, people would begin exploiting them in new ways, some clever, some clearly candidates for a list of what not to do. The compiler itself used 1.5 features, so we gained direct experience in maintaining 1.5 code as well.

As we mentioned, this is not an introductory article. You should know roughly what the new features are, and we'll talk about some of the interesting, hopefully non-obvious implications and uses. These tips are a somewhat random collection of things we ran into, loosely grouped by language feature.

We'll start with the simplest features and work our way toward the most advanced ones. Generics is an especially rich subject and occupies about half of this article.

For-Each Loop

The new for-each loop provides a simple, consistent syntax for iterating over collections and arrays. There are just a couple of interesting items to mention.

Init Expression

The initialization expression is evaluated only once inside the loop. This means that you can often remove a variable declaration. In this example, we had to create an integer array in order to hold the results of computeNumbers() to prevent reevaluation of that method on each pass through the loop. You can see the bottom code is a little cleaner than the above, and doesn't leak the variable "numbers."

Without For Each:

```
int sum = 0;
Integer[] numbers = computeNumbers();
for (int i=0; i < numbers.length ; i++)
 sum += numbers[i];
```

With:

```
int sum = 0;
for ( int number: computeNumbers() )
 sum += number;
```

Limitations

Sometimes you need access to the iterator or index during iteration. Intuitively it seems like the for-each loop should allow this. It doesn't. Take the following example:

```
for (int i=0; i < numbers.length ; i++) {
  if (i != 0) System.out.print(",");
  System.out.print(numbers[i]);
}
```

We want to print out a comma-separated list of the values in the array. We need to know whether we're on the first item in order to know if we should print a comma. With for-each, there's no way to get at this info. We'd need to keep an index ourselves, or a boolean indicating whether or not we're past the first item.

Here's another example:

```
for (Iterator<Integer> it = n.iterator() ; it.hasNext() ; )
 if (it.next() < 0)
  it.remove();
```

In this case, we want to remove the negative items from a collection of integers. In order to do this, we need to call a method on the iterator, but when using for-each, this iterator would be hidden from us. Instead, we need to just use the pre-1.5 method of iterating.

Note, by the way, that Iterator is generic, so the declaration is Iterator<Integer>. A lot of people seem to miss this and use Iterator in its raw form.

Annotations

Annotation processing is a very large topic. We're not going to cover all of the possibilities and pitfalls of it, as we're limiting our article to core language features.

We will, however, discuss the built-in annotations and the limitations of annotation processing in general.

Jess Garms works at BEA Systems.

jess.garms@bea.com

Tim Hanson works at BEA Systems.

tim.hanson@bea.com

Suppress Warnings

This annotation turns off compiler warnings at a class or method level. Sometimes you know better than the compiler that your code must use a deprecated method, or perform some action that cannot be statically determined to be type-safe, but in fact is.

```
@SuppressWarnings("deprecation")
public static void selfDestruct() {
 Thread.currentThread().stop();
}
```

This is probably the most useful of the built-in annotations. Unfortunately, javac doesn't support it as of 1.5.0_03. It is supported in 1.6, however, and Sun is working on back-porting it to 1.5.

It is supported in Eclipse 3.1, and possibly other IDEs as well. This allows you to keep your code entirely warning-free. If a warning shows up when compiling, you can be certain that you just added it – helping to keep you aware of possibly unsafe code. With the addition of generics, this is even more desirable.

Deprecated

Unfortunately, this one is a little less useful. It's meant to replace the @deprecated javadoc tag, but since it doesn't have any fields, there's no way to suggest to the user of a deprecated class or method what they should be using as a replacement. Most uses will require both the Javadoc tag and this annotation.

Override

This indicates that the method it annotates should be overriding a method with the same signature in a superclass.

```
@Override
public int hashCode() {
    …
}
```

Take the above example – if you were to fail to capitalize the "C" in hashCode, you wouldn't get an error at compile time but, at runtime, your method would not be called as you expected. By adding the Override tag, the compiler will complain if it doesn't actually perform an override.

This also helps in the case where the superclass changes. If, say, a new parameter were added to this method and the method itself renamed, the subclass will suddenly fail to compile, as it no longer overrides anything in the super.

Other Annotations

Annotations can be extremely useful in other situations as well. They work best for frameworks like EJB and Web services, when behavior is not directly modified but rather enhanced, especially in the case of adding boiler-plate code.

Annotations cannot be used as a preprocessor. Sun's design specifically precludes modifying the byte code of a class directly because of an annotation. This is so that the results of the language can be properly understood and tools like IDEs can perform deep code analysis and functions like refactoring.

Annotations are not a silver bullet. When first running across them, people are tempted to try all sorts of tricks. Take this next proposal we got from someone:

```
public class Foo {

  @Property
  private int bar;

}
```

The idea here was to automatically create getter and setter methods for the private field "bar." Unfortunately, this is a bad idea for two reasons: (1) it doesn't work, and (2) it makes this code harder to read and deal with.

It can't be done because as we mentioned, Sun specifically precludes modifying the class that an annotation appears in.

Even if it were possible, it's not a good idea because it makes this code harder to understand. Someone looking at this code for the first time will have no idea that this annotation creates methods. Also, if in the future you need to do something inside one of those methods the annotation is useless.

In summary, don't try to use annotations to do what regular code would do.

Enumerations

Enums are a lot like public static final ints that have been used for many years as enum values. The biggest and most obvious improvement over ints is type safety – you cannot mistakenly use one type of enum in place of another, unlike ints, which all look the same to the compiler. With very few exceptions, you should replace all enum-like int constructs with enum instances.

Enums offer a few additional features as well. There are two utility classes, EnumMap and EnumSet, which are implementations of standard collections optimized specifically for enums. If you know your collection will contain only enums, you should use these specific collections instead of HashMap or HashSet.

For the most part you can do a drop-in replacement of any public static final ints in your code with enums. They're comparable, and can be statically imported so that references to them look identical – even in the case of an inner class (or inner enum). Note that when comparing enums, the order they are declared indicates their ordinal value.

"Hidden" Static Methods

There are two static methods that appear on all enum declarations that you write. They don't appear in the javadoc for java.lang.Enum, as they are static methods on enum subclasses, not on Enum itself.

The first, values(), returns an array of all of the possible values for an enum.

The second, valueOf(), returns an enum for the provided string, which must match the source code declaration exactly.

Methods

This is one of our favorite aspects of enums: they can have methods. In the past you might have some code that performed a switch on a public static final int in order to translate from a database type into a JDBC URL. Now, you can have

a method directly on the enum itself, which can clean up code dramatically. Here's an example of how this is done, with an abstract method on the DatabaseType enum and implementations provided in each enum instance:

```
public enum DatabaseType {
    ORACLE {
        public String getJdbcUrl() {…}
    },
    MYSQL {
        public String getJdbcUrl() {…}
    };
    public abstract String getJdbcUrl();
}
```

Now your enum can provide its utility method directly. For instance:

```
DatabaseType dbType = …;
String jdbcURL = dbType.getJdbcUrl();
```

Previously you would have had to know where the utility method was for getting the url.

Varargs

Varargs can really clean up some ugly code, when used correctly. Here's the canonical example: a log method that takes a variable number of string arguments.

```
Log.log(String code)
Log.log(String code, String arg)
Log.log(String code, String arg1, String arg2)
Log.log(String code, String[] args)
```

The interesting item to discuss about varags is the compatibility if you replace the first four examples with a new, vararged one:

```
Log.log(String code, String… args)
```

All of them are source compatible. That is, if you recompile all callers of the log() method, you can just replace all four methods directly. If, however, you need backward binary compatibility, you'll need to leave in the first three. Only the final method, taking an array of strings, is equivalent to, and thus can be replaced by, the vararged version.

Casting

You should avoid casting with varargs in cases where you simply expect the caller to know what the types should be. Take this example, where the first item is expected to be a string, and the second an exception:

```
Log.log(Object… objects) {
    String message = (String)objects[0];
    if (objects.length > 1) {
      Exception e = (Exception)objects[1];
      // Do something with the exception
    }
}
```

Instead, your method signature should be like the following, with the string and exception declared separately from the vararg parameter:

```
Log.log(String message, Exception e, Object… objects) {…}
```

Don't try to be too clever. Don't use varargs to subvert the type system. If you need strong typing, use it. PrintStream.printf() is one interesting exception to this rule: it provides type information as its first argument so that it can accept those types later.

Covariant Returns

The primary use of covariant returns is to avoid casts when an implementation's return type is known to be more specific than the API's. In this example, we've got a Zoo interface that returns an Animal object. Our implementation returns an AnimalImpl object, but before JDK 1.5 it had to be declared to return an Animal object:

```
public interface Zoo {
 public Animal getAnimal();
}

public class ZooImpl implements Zoo {
 public Animal getAnimal(){
  return new AnimalImpl();
 }
}
```

The use of covariant returns replaces three anti-patterns:

1. Direct field access. In order to get around the API restriction, some implementations would expose the subclass directly as a field:

```
ZooImpl._animal
```

2. An additional form was to perform the downcast in the caller, knowing that the implementation was really this specific subclass:

```
((AnimalImpl)ZooImpl.getAnimal()).implMethod();
```

3. The last form I've seen is a special method that avoids the problem by coming up with a different signature entirely:

```
ZooImpl._getAnimal();
```

All of these have their problems and limitations. Either they're ugly or expose implementation details that should not be necessary.

With Covariance

The covariant return pattern is cleaner, safer, and easier to maintain. No casts or special methods or fields are required:

```
public AnimalImpl getAnimal(){
 return new AnimalImpl();
}
```

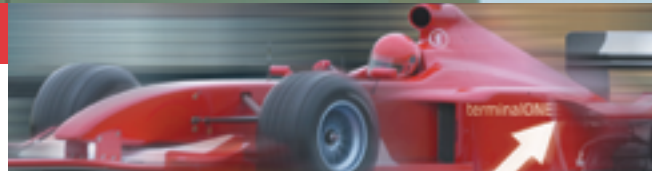
Ah, remember when EDI was young and full of promise?

The future of EDI is B2B over IP

EDI sure had promise in the sixties – but its complexity, inflexibility, and the bottleneck of the VAN make it very costly today.

Simple to integrate, easy to manage, and blazingly fast, **terminalONE** is *the* B2B over IP platform. It intelligently transports, transforms, and routes all your data transactions.

We're about ebusiness adaptability: your business world changes fast – wouldn't it be nice if your data interchange adapted quickly and painlessly along with it?



terminalONE™

Secure, intelligent ebusiness transactions

*high-**velocity***
*high-**volume***
*high-**availability***

Take **terminalONE** for a test drive – ***today***
www.xenos.com/VAN

1 888 242 0695
1 905 763 4468
terminalONE@xenos.com

xenos

Using the result:

```
ZooImpl.getAnimal().implMethod();
```

Generics

Generics is split into using generics and writing generics. These include both generic types and methods. We're not going to talk about the obvious use of List, Set, and Map. Suffice it to say that generic collections are great and should always be used.

We are going to cover using generic methods and how the compiler infers the types. Usually this will just work for you, but when it doesn't the error messages are fairly inscrutable and you will need to know how to fix the problem.

Generic Methods

In addition to generic types, 1.5 introduced generic methods. In this example from java.util.Collections, a singleton list is constructed. The element type of the new list is inferred based on the type of the object passed into the method:

```
static <T> List<T> Collections.singletonList(T o)
```

Example usage:

```
public List<Integer> getListOfOne() {
 return Collections.singletonList(1);
}
```

In the example usage, we pass in an int. The return type of the method is then List<Integer>. The compiler infers Integer for T. This is different from generic types because you don't generally need to explicitly specify the type argument.

This also shows interaction of autoboxing with generics. Type arguments must be reference types, that's why we get List<Integer> and not List<int>.

Generic Methods Without Parameters

The emptyList() method was introduced with generics as a type-safe replacement for the EMPTY_LIST field:

```
static <T> List<T> Collections.emptyList()
```

Example usage:

```
public List<Integer> getNoIntegers() {
 return Collections.emptyList();
}
```

Unlike the previous example, this one has no parameters, so how does the compiler infer the type for T? Basically, it will try once using the parameters. If that does nothing, it tries again using the return or assignment type. In this case, we are returning List<Integer>, so T is inferred to be Integer.

What if you are invoking a generic method in a place other than in a return statement or assignment statement? Then the compiler is unable to do the second pass of type inferencing. In this example, emptyList() is invoked from within the conditional operator:

```
public List<Integer> getNoIntegers() {
 return x ? Collections.emptyList() : null;
}
```

The compiler cannot see the return context, and cannot infer T, so it would give up and assume Object. You would see an error message like "cannot convert List<Object> to List<Integer>".

To fix this, you explicitly pass the type argument to the method invocation. Then the compiler won't try to infer the type arguments for you, and you get the right result:

```
return x ? Collections.<Integer>emptyList():null;
```

The other place where this will happen frequently is in method invocation. If a method takes a List<String> and you try to call this passing emptyList() for that param, you would also need to use this syntax.

Beyond Collections

Here are three examples of generic types that are not collections that use generics in a novel way. All of these come from the standard Java libraries.

- ***Class<T>:*** Class is parameterized on the type of the class. This make it possible to construct a newInstance without casting.
- ***Comparable<T>:*** Comparable is parameterized by the actual comparison type. This provides stronger typing on compareTo() invocations. For example, String implements Comparable<String>. Invoking compareTo() on anything other than a String will fail at compile time.
- ***Enum<E extends Enum<E>>:*** Enum is parameterized by the enum type. An enum called Color would extend Enum<Color>. The getDeclaringClass returns the class object for the enum type. It's different from getClass(), which may return an anonymous class.

Wildcards

The most complex part of generics is understanding wildcards. We'll cover the three types of wildcards and why you might want to use them.

First let's look at how arrays work. You can assign a Number[] from an Integer[]. If you attempt to write a Float into the Number[], it will compile but fail at runtime with an ArrayStoreException:

```
Integer[] ia = new Integer[5];
Number[] na = ia;
na[0] = 0.5; // compiles, but fails at runtime
```

If we try to translate that example directly into generics, it fails at compile time because the assignment isn't allowed:

```
List<Integer> iList = new ArrayList<Integer>();
List<Number> nList = iList; // not allowed
nList.add(0.5);
```

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

>>> Think JReport®

Deliver Intuitive, Easy-to-Understand Reports

- Distribute interactive reports via any Web browser with JReport's 100% DHTML client; no ActiveX or other client downloads needed to support filter, sort, search and drill.
- Reduce the report maintenance burden on developers since users can customize their reports.
- Cascading parameters provide users with run-time flexibility.

Seamlessly Integrate with Your Java Infrastructure

- 100% pure Java architecture, J2EE-compliant and a Swing-based report design tool to leverage your existing Integrated Development Environment (IDE).
- APIs exposed to support integration into your application.
- Modular and re-usable components enable you to embed customized reporting in your application.

Robust Security for Enterprise Deployment

- Security permissions controlled by groups, roles, realms, or users to row-, column- and cell-level for each report.
- Synchronization with external sources allows you to leverage existing security methods like LDAP, Active Directory, Lotus Domino and Novell Directory Server.
- JReport Security API for complete integration with existing security systems.

Support Enterprise Service Level Requirements

- Clustering, failover and load balancing.
- Scales from single-CPU to large, multi-CPU and clustered server environments.
- Report deployment and management with on-demand report viewing, scheduling and version control.
- Export reports from a single template into a variety of formats including DHTML, HTML, PDF, Excel, RTF and CSV.

Forcing a Windows-based reporting tool into your J2EE environment is like putting a square peg into a round hole. You can do it, but why go through the pain for something that doesn't fit.

Only JReport® is a 100% pure Java-based enterprise reporting solution that meets all of your end-user requirements today and provides a stable, standards-based J2EE platform for the future.

See for yourself how easily you can embed our enterprise-class reporting engine into your Java application.

Download a FREE, fully functional copy of JReport at www.jinfonet.com/jp12.htm or call (301) 838-5560.

JReport®
JINFONET SOFTWARE

With generics, you will never get runtime ClassCastExceptions as long as you have code that compiles without warnings.

Upper Bounded Wildcards

What we want is a list whose exact element type is unknown.

- A List<Number> is a list whose element type is the concrete type Number – exactly.
- A List<? extends Number> is a list whose exact element type is unknown. It is Number or a subtype.

Upper Bounds

If we update our original example, and assign to a List<? extends Number>, the assignment now succeeds:

```
List<Integer> iList = new ArrayList<Integer>();
List<? extends Number> nList = iList;
Number n = nList.get(0);
nList.add(0.5); // Not allowed
```

We can get Numbers out of the list because no matter what the exact element type of the list is (Float, Integer, or Number), we can still assign it to Number.

We still can't insert floats into the list. This fails at compile time because we can't prove that this is safe. If we were to add a float into the list, it would violate the original type safety of iList – that it stores only Integers.

Wildcards give us more expressive power than is possible with arrays.

Why Use Wildcards

In this example, a wildcard is used to hide type information from the user of the API. Internally, the set is stored as CustomerImpl. To the user of the API, all they know is that they are getting a set from which they can read customers.

Wildcards are necessary here because you can't assign from a Set<CustomerImpl> to a Set<Customer>.

```
public class CustomerFactory {

 private Set<CustomerImpl> _customers;

 public Set<? extends Customer> getCustomers() {
     return _customers;
 }

}
```

Wildcards and Covariant Returns

Another common use for wildcards is with covariant returns. The same rules apply to covariant returns as assignment. If you want to return a more specific generic type in an overridden method, the declaring method *must* use wildcards:

```
public interface NumberGenerator {
 public List<? extends Number> generate();
}
public class FibonacciGenerator extends NumberGenerator {
 public List<Integer> generate() {
  …
 }
}
```

If this were to use arrays, the interface could return Number[] and the implementation could return Integer[].

Lower Bounds

We've talked mostly about upper bounded wildcards. There is also a lower bounded wildcard. A List<? super Number> is a list whose exact "element type" is unknown, but it is number or a super type of number. So it could be a List<Number> or a List<Object>.

Lower bounded wildcards are not nearly as common as upper bounded wildcards. But when you need them, they are essential.

Lower vs Upper Bounds

```
List<? extends Number> readList = new ArrayList<Integer>();
Number n = readList.get(0);

List<? super Number> writeList = new ArrayList<Object>();
writeList.add(new Integer(5));
```

The first list is a list that you can *read* numbers from.

The second list is a list that you can *write* numbers to.

Unbounded Wildcard

Finally, the List<?> is a list of anything. Almost the same as List<? Extends Object>. You can always read Objects, but you cannot write to the list.

Wildcards in Public APIs

To summarize, wildcards are great for hiding implementation details from callers as we saw a few slides back, but even though lower bounded wildcards appear to provide read-only access, they do not due to non-generic methods like remove(int position). If you want a truly immutable collection, use the unmodifiableCollection(), etc.

Be aware of wildcards when writing APIs. In general, you should try to use wildcards when passing generic types. It makes the API accessible to a wider range of callers.

In this example, by accepting a List<? extends Number> instead of List<Number>, the method can be called by many different types of lists:

```
void removeNegatives(List<? extends Number> list);
```

Constructing Generic Types

Now we'll cover constructing your own generic types. I'll show example idioms where type safety can be improved by using generics, as well as common problems that occur when trying to implement generic types.

Collection-Like Functions

This first example of a generic class is a collection-like ex-

ample. Pair has two type parameters, and the fields are instances of the types:

```
public final class Pair<A,B> {
 public final A first;
 public final B second;

 public Pair(A first, B second) {
  this.first = first;
  this.second = second;
 }
}
```

This makes it possible to return two items from a method without having to write special-purpose classes for each two-type combo. The other thing you could have done is return Object[], which isn't type-safe or pretty.

In the usage here, we return a File and a Boolean from a method. The client of the method can use the fields directly without casting:

```
public Pair<File,Boolean> getFileAndWriteStatus(String path){
 // create file and status
 return new Pair<File,Boolean>(file, status);
}

Pair<File,Boolean> result = getFileAndWriteStatus("…");
File f = result.first;
boolean writeable = result.second;
```

Beyond Collections

Here is an example where generics are used for additional compile-time safety. By parameterizing the DBFactory class by the type of Peer it creates, you are forcing Factories to return a specific subtype of Peer:

```
public abstract class DBFactory<T extends DBPeer> {
    protected abstract T createEmptyPeer();

    public List<T> get(String constraint) {
        List<T> peers = new ArrayList<T>();
        // database magic
        return peers;
    }
}
```

By implementing DBFactory<Customer>, the CustomerFactory is forced to return a Customer from createEmptyPeer:

```
public class CustomerFactory extends DBFactory<Customer>{

    public Customer createEmptyPeer() {
        return new Customer();
    }
}
```

Generic Methods

Whenever you want to place constraints on a generic type between parameters or a parameter and a return type, you probably want to use a generic method.

For example, if you write a reverse function that reverses in place, you don't need a generic method. However, if you want reverse to return a new list, you'd like the element type of the new list to be the same as the list that was passed in. In that case, you need a generic method:

```
<T> List<T> reverse(List<T> list)
```

Reification

When implementing a generic class, you may want to construct an array, T[]. Because generics is implemented by erasure, this is not allowed.

You might try to cast an Object[] to T[]. This is *not* safe.

Reification – Solution

The solution, courtesy of the generics tutorial, is to use a "Type Token". By adding a Class<T> parameter to the constructor, you force clients to supply the correct class Object for the type parameter of the class:

```
public class ArrayExample<T> {
    private Class<T> clazz;

    public ArrayExample(Class<T> clazz) {
        this.clazz = clazz;
    }

    public T[] getArray(int size) {
        return (T[])Array.newInstance(clazz, size);
    }
}
```

To construct an ArrayExample<String>, the client would have to pass String.class to the constructor because the type of String.class is Class<String>.

Having the class objects makes it possible then to construct an array with exactly the right element type.

Migrating to Generics

Finally, we'll just briefly talk about migrating 1.4 code to using Generics:

- You can generify existing classes and interfaces.
- You can convert raw collection uses to generic collections, even in public methods. This will not break clients who override.
- If you need to pass a collection to a 1.4-level library, you can use checked collections. These will fail fast if that library attempts to put something in a collection that shouldn't belong. This is an interesting example of the "Type Token" pattern we mentioned.
- In general, if you want to know if something is safe, look at the standard libraries. There are plenty of examples of migrating classes while maintaining source and binary compatibility.

Conclusion

In summary, the new language features make for a substantial change to Java. By understanding when and how to use them, you'll write better code.

Proxy Cache

A practical implementation

by Justin Knowlden

A proxy cache is a behavioral technique that provides the ability to cache a result-set (output) from a service call via a proxy using the argument-set (input) as the cache key. Through this proxy, all calls to any concrete instances of a defined type, service, are made. In this proxy, input is used as the key to obtain from or provide to a cache the output from a called service. If output has already been cached for a given input, the proxy returns the cached output without forwarding the call to the actual service.

This article presents a software design approach that will allow you to add a transparent caching layer to your applications. This approach will work for all types of applications (middle-tier, client-side, batch, etc.) under many scenarios (Web services, database access, etc.).

Although this article does refer to caching, it's not about caching. Instead, this article attempts to provide a description of how to incorporate a feature, such as caching, modularly and without major impact to existing code.

Provided first is a description of how my colleagues and I ended up at a proxy cache and is followed by a generic summary of how to design and implement a proxy cache solution.

Justin Knowlden is a solutions architect with United Airlines. Prior to United, he helped develop the MyPoints.com product from its inception. Justin is actively participating in the open source community (see Helium and ESP). When not programming he is a husband, a basketball and football coach for his son, and an environmental and animal rights activist.

justin.knowlden@united.com

Background

While working with several colleagues on a recent project, we decided to avoid any performance problems that were caused by repeatedly retrieving data from our back-end data store and from the performance problems related to object creation when generating responses from the services in our service layer. To accomplish this, we needed to incorporate a cache mechanism into that service layer.

The service layer was fairly simple in that most of the services were read-only with some basic business logic applied behind the scenes. We were not using any object/relational (O/R) mapping tools, just basic Data Access Objects (DAOs) and, therefore, several ideas were tossed around.

One idea was to simply use an O/R tool. Any good O/R tool would include a caching mechanism to avoid hitting the database unnecessarily. However, we decided against this for a couple of reasons:

- Previous and bad experiences with a commercial O/R tool specifically related to caching and synchronization.
- A motivating drive to keep things as simple as can be, but no simpler.
- It did not solve the problem related to object creation.

Another idea was to utilize a third-party caching solution. Although there are many to choose from, they did not address all of our software design concerns. We therefore realized that we needed to define some objectives for our caching system. These objectives are defined below:

- The caching system should be transparent to the user and the used.
- The caching system should eliminate the sprinkling of cache code throughout the code base, i.e., to implement caching in a single place so as not to violate the Single Responsibility Principle (SRP) and Once And Only Once (OAOO) concept.

In addition, many of the third-party systems were fairly heavyweight, requiring more configuration than we desired and a fair amount of physical space for the library or libraries. We were not strictly opposed to using these tools, but we needed something else first.

Another viable option for us was to introduce caching code into each service, but this was not suitable either as it violated some of our principles. The SRP was violated when the service would have been responsible for its normal, expected behavior and also caching.

OAOO was violated when we would have had to introduce the same code into each service. We could have abstracted the behavior into a base class or supporting utility class, but SRP would still have been violated, so why bother.

Proxy Cache

After some consideration, we realized that we were saying we wanted an aspect of caching applied to each of our services. The role of this aspect would be to (1) check a cache to determine if the service had already been called with the provided input and if so, would (2) return the cached output or else it would (3) complete the call and then cache the output.

Java does not have extensive built-in support for Aspect-Oriented Programming, and we did not want to introduce an AOP extension such as AspectJ. However, Java does provide built-in support for proxies via the dynamic proxy class (proxy class), provided by the standard Java platform since JDK1.3. A proxy class could be used to encapsulate our needed behavior without violating our core principles and making us feel all "dirty." From this idea was born our concept of a proxy cache.

Note: The choice to not use an O/R mapping tool does not negate using one in the future. You could potentially use a proxy cache and an O/R mapping tool concurrently.

Dynamic Proxy

At the heart of the proxy cache technique is the Proxy pattern. If you are not familiar with what the Proxy Pattern is, the accepted definition is that a proxy "[provides] a surrogate or placeholder for another object to control access to it." The goal of the proxy cache is to be a surrogate to the service, but with a small caveat that it must also be able to present itself as the actual service, such that the caller of the service cannot differentiate the two. The proxy cache for a given service must be able to stand-in for the service itself.

In Java, there are obviously many ways that a proxy can be implemented, but this article will utilize the well-established proxy class, java.lang.reflect.Proxy. Using a proxy class will allow the proxy cache to disguise itself as the service without implementing any user-defined interfaces.

The basic protocol for using the proxy class is as follows:

1. Provide an implementation of java.lang.reflect.InvocationHandler.
2. Generate a new proxy instance by providing a classloader, an array of interfaces the proxy shall implement, and an instance of your invocation handler.
3. Cast the proxy object to a type consistent with the interfaces you provided when generating the proxy.

Below is an example of the implementation of this protocol:

```
InvocationHandler handler =
  new FooInvocationHandler(new Bar());
Foo f = (Foo) Proxy.newProxyInstance(
    Foo.class.getClassLoader(),
    new Class[]{ Foo.class }, handler);
```

In your invocation handler you are required to implement an invoke() method that must accept the proxy instance, a reference to the method that is being called on that instance, and any arguments that are being passed to that instance. In this invoke() method you can provide an epilogue and/or prologue to the actual method call; i.e., you can "wrap" the intended method call with your own custom behavior.

The invocation handler must also have access to a concrete instance of the desired object. This is perhaps the trickiest aspect of the proxy class framework as it requires a priori knowledge of the caller's intentions or a very specific implementation of an invocation handler. In the case of the proxy cache, the most useful implementation is to pass the concrete object as an argument to the constructor of the invocation handler. This, however, implies a bijectional relationship between the proxy and the proxied object, which may only be bad if there is an explosion of objects needing a proxy.

Implementing a Proxy Cache

In general, we would like to limit the number of places where proxy cache is actually referenced; therefore, the most basic way to implement proxy cache is through a factory or builder. A factory is used to isolate the details of generating a new service. Isolating these details is important because our ServiceFactory will be responsible for generating concrete service instances, generating invocation handler (CacheProxy) instances for services, and returning proxies to the services.

Shown in Figure 1 is a simple entity relationship diagram of the framework that will be implemented. Generally, you will already have everything shown in the diagram well established in your system except for CacheProxy. Why? Because you're not going to introduce a proxy cache until you actually need it. Introduce caching only when you know there is a problem. In Figure 1, all interfaces are illustrated as a rounded-rectangle.

As is shown, the Service interface knows only about itself:

```
package service;
public interface Service {
  public Object service(
    Object toService);
}
```

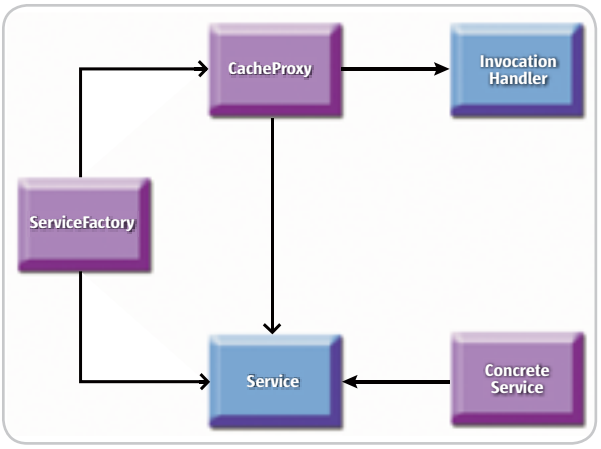


Figure 1 Entity diagram of the service framework

The CacheProxy class knows about the service and is an implementation of the InvocationHandler interface. Finally, the ServiceFactory knows about everything service related (including the Concrete Service, albeit implicitly).

Where does the caching behavior go? The responsibility of caching firmly belongs to CacheProxy. But, how is the CacheProxy created so that it may enable caching? Normally, the ServiceFactory would create a Concrete Service and basically return it. When utilizing proxy cache, ServiceFactory will still create the Concrete Service instance, but will pass that instance to a new instance CacheProxy and return the CacheProxy instance instead. Therefore, whenever a method is called on the service (even those

not defined in the service interface), the call will be trapped by the CacheProxy, wherein a cache can be investigated prior to passing the call on to the Concrete Service.

In order to implement the rest of the framework we will need to start with a very simple concrete service, which we will call EchoService. The EchoService implements the service interface and simply returns the request object as the response object; or, EchoService echoes the input as its output. This simple construct will help illustrate caching functionality when we actually implement it.

```
package service;
public class EchoService
implements Service {
  public Object service(
  Object toService) {
    return toService;
  }
}
```

Listing 1 shows the implementation of a ServiceFactory that only knows how to create an EchoService. In the method newEchoService(), notice that even though a new EchoService instance is created, it is wrapped in a new CacheProxy instance and returned as a dynamic proxy. While the reflection taking place here might appear to be a heavy process, it is essentially a one-time hit because the Proxy class internally caches Proxy class definitions given the particular set of arguments to Proxy.newProxyInstance(). We could also create the Proxy class ahead of time since we expect to only return objects of type service, but the code is left this way for the sake of brevity; therefore, it is up to you to refactor this code.

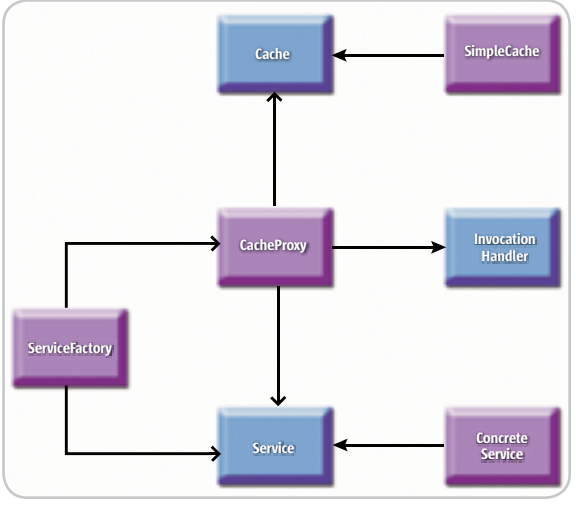


Figure 2 Final entity relationship diagram

Listing 2 shows our Proxy Cache implementation, aptly named CacheProxy. Even though CacheProxy is an implementation of an InvocationHandler, the suffix –Proxy is left in to illustrate that CacheProxy contains our actual proxy code.

Notice that our CacheProxy is somewhat light in terms of desired functionality. In fact, it's quite useless. If we were to throw this code into production, EchoService would return the request object each and every time. From a black-box perspective, we actually desire that the same output be provided for a given input. However, if we can imagine that EchoService is actually a fairly complex service that performs several time-consuming operations on its input, we quickly realize that it would be better to just return a cached response ahead of time without enduring the performance hit.

Refactoring the Proxy Cache

There are many ways to incorporate a caching mechanism into our CacheProxy class. The most basic approach is to use a java.util.Map, more specifically, a HashMap. This solution sounds simple, but the problem with this approach is made visible in this way: if we initialize the Map instance directly in the CacheProxy, the Map will only survive for as long as the CacheProxy instance does. Because the ServiceFactory creates new CacheProxy instances every time a service is requested, we will, in effect, never cache anything.

We don't want to inject the Map into CacheProxy because only the CacheProxy should care about the Map. Therefore, a suitable solution to this problem is to have the ServiceFactory manage the instances of proxies that it creates. There are, however, several side-effects to this approach. For starters, the ServiceFactory is now acting more like a resource manager than a factory. Although we would normally rename ServiceFactory to something like ServiceManager, at this point, we opt to leave the name as is for the rest of this article.

Second, each CacheProxy instance is a kind of pseudo-singleton. For example, there will be only one EchoService proxy instance per instance of the ServiceFactory. These proxies are only pseudo-singletons because the ServiceFactory is not itself a singleton; meaning, each instance of ServiceFactory will create its own CacheProxy instances. Or better yet, assuming we will only create one ServiceFactory instance per virtual machine, we have what Uncle Bob (Bob Martin) would call "Just Create One," making ServiceFactory a glorified Singulizer.

To ensure one instance of a service per virtual machine, we privately define a class instance variable named *echoService* and lazily initialize it at runtime. The following code snippet can be applied to ServiceFactory to enable lazy-initialization of an EchoService instance:

```
...
private Service echoService;
public synchronized Service newEchoService() {
  if (echoService == null)
    echoService = //create EchoService proxy
  return echoService;
}
...
```

We had to synchronize newEchoService() to ensure that only one instance is ever created lazily for the specific ServiceFactory instance. If newEchoService() is not synchronized in any way, a race condition could exist whereby multiple, simultaneous calls to newEchoService() would produce multiple EchoService instances, thereby reducing the probability of generating a unique cache. This race condition only exists when EchoService is being lazily instantiated.

In addition, if ServiceFactory were to manage more service initializations, we would need a class instance variable per factory method.

Is the Map problem solved? No; how will CacheProxy use the Map? If we want CacheProxy to return the same output for a given input, we should use the argument array provided to invoke() as a key and the response from the call to the real service as the value. Thus, whenever the same input is provided, assuming immutable types or the exact same object instances are being passed, we can retrieve the corresponding output from the Map.

That sounds great too, but Cache-Proxy's generic behavior elicits yet another problem. What happens when two methods of a service (or any object being proxied) share the same method signature? In this case, a call to one method may actually return a previously cached response of another method. The quick-and-easy solution is to keep a cache for each method, which is very feasible considering that the invoke() method provides us with an instance of the Method object that was being called. But, this enhancement creates another headache for us because we need to store the mapping of methods to their caches.

Assuming we have incorporated the

enhancement, we now have a two-level caching scheme using Maps. Are we ready to go yet? In theory, yes, but there is one more item we should address. Because we don't need all of the functionality provided by a Map and we really just want to deal with a cache, we should define a cache type.

```
package service;
public interface Cache {
  public Object retrieve(
       Object key);
  public void store(
       Object key, Object value);
}
```

In our cache, the only behavior we are concerned with is the ability to store and retrieve values via their keys. In addition, if we need any other special behavior for our cache(s), we can implement that behavior in cache, the proper place, and not make CacheProxy responsible for it. Once cache is defined, we can implement a SimpleCache that uses HashMap internally (see Listing 3).

From an object-oriented perspective, realizing the cache type is actually very important. Earlier in this article I stated that CacheProxy is responsible for all caching behavior, which is true, but only to a point. The cache type highlights where the statement breaks down. If we leave all caching behavior in CacheProxy, CacheProxy is then responsible for the investigating the cache and possibly calling the real service (proxy behavior) and managing the actual cache. To keep things simple, we only really want CacheProxy to be responsible for one of those two, therefore, cache is responsible for managing the cache and CacheProxy only needs to know what cache provides.

Now are we ready to move on? Quite so! We finally move on to adding cache behavior to CacheProxy (see Listing 4). The new version of CacheProxy is slightly longer, but not much more. There is also some duplication in the invoke() and findMethodCache() methods related to storing values when a retrieved value is null, but, we'll leave this refactoring for another day.

At the heart of CacheProxy is the invoke() method. In the invoke() method a specific protocol is followed. First, the cache for the method being called is identified and retrieved. This functionality is mostly implemented in the findMethodCache() method, wherein, a new cache instance is created if no cache instance has yet been created for the particular method. Thus, the method instance is itself the key to a cache.

Once the method's cache has been identified, another key is created from the array of arguments that is to eventually be passed to the targeted method. This key is generated via the generateArgumentKey() method. After creating this composite key, the key is used to find any previously cached value from the method's cache retrieved earlier. If no value was previously cached, the proxied object's method is called reflectively (identified by the Method object) and the results of that call are cached. Finally, the cached results are returned back to the

original caller.

It is important to note what is going on in the generateArgumentKey() method. Superficially, the array of arguments is being morphed into a list of arguments (java.util.List). If the array of arguments is null, a predefined *NullKey* is used as the new key. But why convert the array to a list? Lists that extend AbstractList have a unique feature in that the hash-code is generated based on the set of elements in the list, such that two distinct lists containing the exact same elements will produce the exact same hash-code.

On the other hand, two arrays of arguments containing the same elements will produce two distinct hash-codes, inhibiting us from being able to retrieve a cached value given the same set of arguments. In addition, if we were to simply use the object array provided to the invoke() method as the key, we would have an ever-growing cache that never finds anything we are looking for.

It could also be argued that because the hash-code issue is cache related, the job of converting the array to a list belongs to cache. It could be argued, but, the reality is that CacheProxy is actually concerned about presenting identical keys, not cache. Therefore, CacheProxy is responsible for the conversion.

Currently, java.util.Arrays.asList() creates a list that is an extension of AbstractList. If at some point it does not, you could simply create the list yourself (e.g., java.util.ArrayList) and copy the element references from the array.

Applying Advanced Cache Techniques

So far, a very simple cache replacement mechanism has been assumed, as in, cache-forever. But, to realize the full potential of proxy cache we want the data in the cache to be replaced, or at least purged, after some period of time.

Replacement is generally triggered via some event that is either bound temporarily or spatially. Meaning, either a predetermined amount of time has passed or a preallocated size boundary has been exceeded. Some of the most common techniques applied are: time-to-live (TTL), least-recently used (LRU), and least-frequently-used (LFU). It is out of the scope of this article to describe what each of these algorithms means, although, you can probably guess from their names.

However, if we wanted to apply one of these algorithms, we could simply implement another type of cache. For instance, if we wanted to utilize a TTL cache for all of our services that would remove expired entries after 10 minutes (600 seconds), the shell of a TimeToLiveCache would look similar to that shown in Listing 5.

Remember that the proxy cache technique works especially well when the output of a given operation can be guaranteed to be the same for every call to a service or set of services given some predetermined temporal or spatial boundary. Therefore, whatever solution you use, make sure that it does not negate the positive aspects of actually using a cache.

Unit Testing a Proxy Cache

Unit testing is an essential practice in the development of all software, no matter the size or complexity.

All code provided here was developed using the Test-Driven Development methodology (TDD), with the exception of the TimeToLiveCache since there is no real implementation provided. All test cases and their supporting test code have been provided in Listing 6. (Listing 6 and additional source code can be downloaded from http://jdj.sys-con.com.)

In the test code you'll find only five test cases asserting that:

1. EchoService returns the same object that was provided to it.
2. ServiceFactory returns only one instance of a proxy per service.
3. CacheProxy actually caches.
4. CacheProxy can handle a null reference as the set of arguments.
5. CacheProxy differentiates its caches by method name.

For a few of the tests a mock object is used in place of a "real" service that would normally be passed to CacheProxy. Using this mock object allows us to isolate CacheProxy in our tests because, hopefully, the mock object contains very little behavior. Most mock objects should contain only enough logic to record events for investigation later.

There could be one or two tests missing from the provided test cases and there is certainly some duplication in the test code, but, for the most part, the tests provided are enough to cover most basic implementations of proxy cache. In addition, if you are not already practicing the TDD methodology, the examples provided can provide you with more proof of why it is so invaluable. If for no other reason, TDD was invaluable in developing the proxy cache because we didn't have to guess at how we should actually implement the framework.

History

At its core, the proxy cache concept is not new. The concept is borrowed from the memoization technique (meaning "to put in memory"), recognized by Donald Mitchie in his paper "Memo functions and machine learning" from a *Nature* magazine published in 1968.

The proxy cache is an abstraction of the memoization technique and differs from memoization by assuming that the data is likely to change, whereas memoization relies on the data not changing.

Conclusion

The design and implementation of the proxy cache approach provided in this article is simplistic. In the solution my colleagues and I eventually put together, we introduced a synchronization scenario so that all caches running in our application servers could purge themselves when necessary. There is no theoretical limit to what can be introduced into this layer.

In most instances, however, an implementation of the proxy cache that utilizes a basic replacement algorithm should be enough to satisfy any application. Remember that the goal of the proxy cache is to keep your application simple, modular, and free from responsibility creep.

References

- Portland Pattern Repository; *ProxyPattern*: http://c2.com/cgi/wiki?ProxyPattern; last updated 20050124
- Martin, B., et al; *Singleton vs. Just Create One*: http://butunclebob.com/ArticleS.UncleBob.SingletonVsJustCreateOne
- Wikipedia Users; *Memoization*: http://en.wikipedia.org/wiki/Memoization; last updated 20050708-22:55
- Bob Martin, et al; *Single Responsibility Principle*, http://www.objectmentor.com/resources/articles/srp

Acknowledgments

None of the information provided here would have been possible without the following people: Virgil Bistriceanu, Boris Vaysburg, Aleksey Beregov, Murali Kashaboina, and Brett Neumeier (even though he doesn't know it).

Listing 1: Initial ServiceFactory implementation

```
package service;
import java.lang.reflect.Proxy;

public class ServiceFactory {
  public Service newEchoService() {
    return (Service) Proxy.newProxyInstance(
        Service.class.getClassLoader(),
        new Class[]{Service.class},
        new CacheProxy(new EchoService()));
  }
}
```

Listing 2: Initial CacheProxy implementation

```
package service;
import java.lang.reflect.InvocationHandler;
import java.lang.reflect.Method;

public class CacheProxy implements InvocationHandler {
  private final Object obj;
  public CacheProxy(Object toProxy) {
    this.obj = toProxy;
  }

  public Object invoke(Object proxy, Method method, Object[] args)
  throws Throwable {
    return method.invoke(obj, args);
  }
}
```

Listing 3: SimpleCache, HashMap implementation of a Cache

```
package service;
import java.util.HashMap;
import java.util.Map;

public class SimpleCache implements Cache {
  private Map values;
  public SimpleCache() {
    values = new HashMap(16);
  }

  public Object retrieve(Object key) {
    return values.get(key);
  }

  public void store(Object key, Object value) {
    values.put(key, value);
  }
}
```

Listing 4: Final implementation of CacheProxy

```
package service;
import java.lang.reflect.InvocationHandler;
import java.lang.reflect.Method;
import java.util.Arrays;

public class CacheProxy implements InvocationHandler {
  private static final Object NullKey = new Object();
  private final Object obj;
  private Cache caches;

  public CacheProxy(Object toProxy) {
    this.obj = toProxy;
    caches = new SimpleCache();
  }

  public Object invoke(Object proxy, Method method, Object[] args)
  throws Throwable {
    Cache cache = findMethodCache(method);
    Object key = generateArgumentKey(args);
    Object value = cache.retrieve(key);
    if (value == null)
      cache.store(key, value = method.invoke(obj, args));
    return value;
  }

  private Cache findMethodCache(Method method) {
    Cache cache = (Cache) caches.retrieve(method);
    if (cache == null)
      caches.store(method, cache = new SimpleCache());
    return cache;
  }

  private Object generateArgumentKey(Object[] args) {
    return args != null ? Arrays.asList(args) : NullKey;
  }
}
```

Listing 5: Time-to-live shell implementation

```
package service;

public class TimeToLiveCache
implements Runnable, Cache {
  private final int ttlSeconds;
  private Thread ttlThread;

  public TimeToLiveCache(int ttlSeconds) {
    this.ttlSeconds = ttlSeconds;
    ttlThread = new Thread(this);
    ttlThread.start();
  }

  public void run() {
    /* wake up and remove entries
     * every so often
     */
  }

  public Object retrieve(Object key) {
    /* basic retrieval */
  }

  public void store(Object key, Object value) {
    /* basic storage, but will also need to
     * record the time the entry was added
     */
  }
}
```

JSF and AJAX

Introducing a new open source project

by Jonas Jacobi and John Fallows

In our previous article – "Rich Internet Components with JavaServer Faces" (***JDJ***, Vol. 10, issue 11) – we discussed how JavaServer Faces can fulfill new presentation requirements without sacrificing application developer productivity building Rich Internet Applications (RIA). We discussed how JSF component writers can utilize technologies, such as AJAX and Mozilla XUL, to provide application developers with rich, interactive, and reusable components.

To use AJAX and Mozilla XUL with JSF, component writers have to make sure to provide any resource files needed by these technologies, such as images, style sheets, or scripts. The standard approach to providing resource files for a JSF component library is to serve them directly out of the Web application root file system. These resources are usually packaged in an archive (such as a ZIP file) and shipped separately from the JSF component library.

This article introduces a new open source project – Weblets – which can be found on the java.net Website (http://weblets.dev.java.net). The goal of this open source project is to provide JSF component writers with a facility that can serve resource files out of a Java archive (JAR), rather than serving them from the Web application root file system. Unlike traditional Web applications, which have statically configured URL mappings defined in web.xml, there is a need for dynamic configuration of URL mappings, based on the presence of a component library JAR. In essence, Weblets provide developers with an easy way to package Web application resources in the same Java archive (JAR) that their implementation code resides in.

Jonas Jacobi is a technology evangelist for Oracle's Java/J2EE tool offering, JDeveloper, and over the past three years has been responsible for JavaServer Faces, Oracle ADF Faces, and Oracle ADF Faces Rich Client development features within Oracle JDeveloper. Jonas has been in the software business for 15 years. Prior to joining Oracle, he worked at several software companies in Europe, covering many roles including support, consulting, development, and project team leadership.

jonas.jacobi@oracle.com

Resource Loading

Let's assume that we have a JSF component that needs to have a JavaScript file, myScript.js, served to the client. This JavaScript file is used by the component to provide some level of richness when interacted with by the end user. This JavaScript file is traditionally served by the Web application via a relative path that is hard coded into the actual Renderer code for the JSF component. This requires the application developer to deploy additional resources that are delivered and packaged in a separate archive file, e.g., ZIP, often referred to as "installables."

It is important to note that the JavaServer Faces HTML basic RenderKit does not have any images, styles, or scripts, so there is no standard solution to the Faces resource packaging problem.

The following sample Renderer code illustrates the installables approach to serving a JavaScript file, /myresources/myScript.js, from the Web application root file system.

```
ViewHandler handler = context.getApplication().getViewHandler();
String resourceURL = handler.getResourceURL(context,"/myresources/
                                                    myScript.js");
out.startElement("script", null);
out.writeAttribute("type", "text/javascript", null);
out.writeAttribute("src", resourceURL, null);
out.endElement("script");
```

Although the installables approach is convenient for the JSF component author, it does increase the installation burden on the application developer, who must remember to extract the installables archive each time the component library is upgraded to a new version. Therefore, we need a way to package our additional resources into the same JAR file containing the Renderer classes, simplifying deployment for application developers using our component library.

Using Weblets

The open source Weblets project aims to solve the resource-packaging problem in a generic and extensible way so that it can be leveraged by all JavaServer Faces component writers, while placing only a minimal installation burden on the application developer.

A Weblet acts as a mediator that intercepts requests from the client and uses short Web URLs to serves resources from a JAR file. Unlike the Servlet or Filter approach, a Weblet can be registered and configured inside a JAR, so the component library Renderers, their resource files, and the Weblet configuration file (weblets-config.xml) can all be packaged together in the same JAR. The Weblet Container can be registered just once in the Web application configuration file – web.xml – for all component libraries. There is no need to separately deploy additional installables when the component libraries are upgraded to new versions.

It is important to note that all resources served up by Weblets are *internal* resources, used only by the Renderer. Any resources, like images, that are provided by the application, are supplied as component attribute values and loaded from the context root as external resources.

Weblet Architecture

Although Weblets were designed to be used by any Web client, the Weblets implementation has been integrated with JavaServer Faces using a custom ViewHandler, WebletsViewHandler, as shown in Figure 1. During rendering of the main JavaServer Faces page, the WebletsViewHandler is responsible for converting Weblet-specific resource URLs into the actual URLs used by the browser to request Weblet-managed resources.

After receiving the rendered markup for the main page, the browser downloads each additional resource using a separate request. Each request for a Weblet-managed resource is intercepted by the WebletsPhaseListener, which then asks the WebletContainer to stream the Weblet-managed resource file out of the component library JAR.

The WebletContainer is designed to leverage the browser cache where possible. This improves overall rendering performance by minimizing the total number of requests made for Weblet-managed resource files.

To ensure flexibility and optimization, and avoid collisions with existing Web application resources, Weblets can be configured by application developers to override any default settings provided by the component author.

Using Weblets in a Component library

Weblets are configured using a weblets-config.xml file, which must be stored in the /META-INF directory of the component library JAR. Configuring a Weblet is similar to configuring a Servlet or a Filter. Each Weblet entry in the weblets-config.xml file has a Weblet name, implementation class, and initialization parameters (see Listing 1). The Weblet mapping associates a particular URL pattern with a specific Weblet name, e.g., org.myapp.html. The Weblet name and default URL pattern define the public API for the Weblet-managed resources and should not be modified between releases of your component library, in order to maintain backward compatibility.

Our component library packages resources in the org.myapp.faces.renderer.html.resources Java package and makes them available to the browser using the default URL mapping of /myresources/*.

The PackagedWeblet is a built-in Weblet implementation that can read from a particular Java package using the ClassLoader and stream the result back to the browser. The package initialization parameter tells the PackagedWeblet which Java package to use as a root when resolving Weblet-managed resource requests.

Weblet Versioning

Weblets also has built-in support for versioning of the component library. This is used to allow the browser to cache packaged resources such as myScript.js when possible, preventing unnecessary round-trips to the Web server.

Each time the browser renders a page, it will ensure that all resources used by that page are available. During the initial rendering of the page, the browser populates its cache with the contents of each resource URL by downloading a fresh copy from the Web server. As it does so, the browser records the Last-Modified and Expires timestamps from the response headers. The cached content is said to have expired if the current time is later than the expiration timestamp, or if no expiration timestamp information exists.

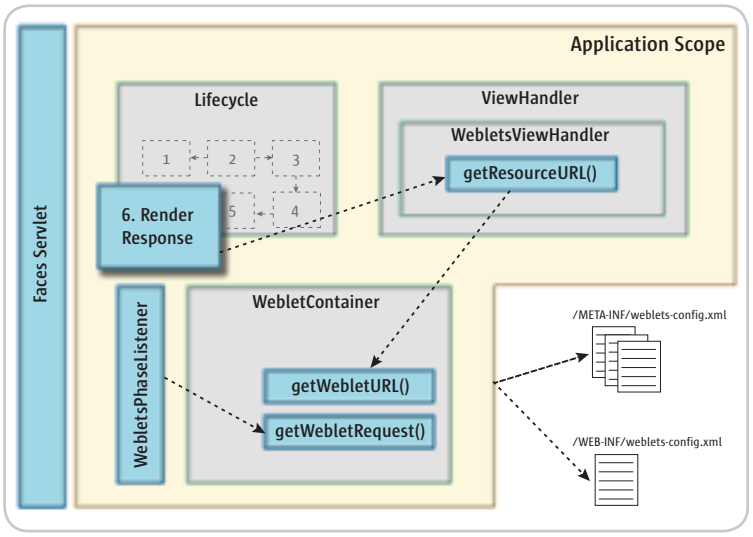


Figure 1 High-overview of Weblet architecture

On the next render of the same page, the browser checks to see if the locally cached resource has expired. The locally cached copy is reused if it has not expired. Otherwise, a new request is made to the Web server, including the last modified information in the If-Modified-Since request header. The Web server responds by either indicating that the browser cache is still up-to-date, or by streaming the new resource contents back to the browser with updated Last-Modified and Expires timestamps in the response headers.

Weblets use versioning to leverage the browser cache behavior so that packaged resources can be downloaded and cached as efficiently as possible. The browser only needs to check for new updates when the cache has been emptied or when the component library has been upgraded at the Web server.

Listing 2 illustrates the Weblets versioning feature by adding a 1.0 version to our org.myapp.html Weblet.

By specifying a Weblet version, you indicate that the packaged resource is not going to change until the version number changes. Therefore, the version number is included as part of the resource URL determined at runtime by the WebletsViewHandler, e.g., /myresources$1.0/myScript.js. When the WebletContainer services this request, it extracts the version number from the URL and determines that the resource should be cached and never expire. As soon as a new version of the component library is deployed to the Web application, the resource URL created at runtime by the WebletsViewHandler changes, e.g., /myresources$2.0/myScript.js, thus the browser's cached copy of myScript.js for version 1.0 is no longer valid because the URL is different.

During development, the contents of packaged resources can change frequently, so it is important for the browser to keep checking back with the Web server to detect the latest resource URL contents. This check happens by default every time the main Web page is rendered if the Weblet version is omitted from weblets-config.xml.

Alternatively the Weblet configuration allows component authors to append -SNAPSHOT to the version number. For example, 1.0-SNAPSHOT, as shown in the following code, indicates that this file is under development and should behave as though the version number has been omitted.

John Fallows is a consulting member of technical staff for server technologies at Oracle Corporation, and has been working in distributed systems for over a decade. During the past five years, he has focused on designing, developing, and evolving Oracle ADF Faces, and is now lead developer for Oracle ADF Faces Rich Client.

john.fallows@oracle.com

```
<?xml version="1.0" encoding="UTF-8" ?>
<weblets-config xmlns="http://weblets.dev.java.net/config" >
  <weblet>
    <weblet-name>org.myapp.html</weblet-name>
    <weblet-class>net.java.dev.weblets.packaged.PackagedWeblet</
      weblet-class>
    <weblet-version>1.0-SNAPSHOT</weblet-version>
    ...
  </weblet>
  ...
</weblets-config>
```

Security

When serving packaged resources from a JAR, extra care must be taken not to make Java class files or other sensitive information accessible by URL. In desktop Java applications, resource files are often stored in a sub-package called "resources" underneath the Java implementation classes that use the resource files. The same strategy is also appropriate for packaged resources in JavaServer Faces component libraries, and has the security benefit of ensuring that only the resource files are accessible by URL. All other contents of the JAR file, including Java implementation classes, are not URL accessible because no Java classes exist in either the "resources" package or in any sub-package of "resources."

Weblets Protocol

Having covered how to configure Weblets, it's time to look at how we can reference resources defined by the Weblet in our renderer. The syntax, defined by the Weblet contract, for returning a proper URL to the JSF page is as follows:

```
<prefix><weblet name><resource>
```

The prefix indicates that this is a Weblet-managed resource, and this is followed by the Weblet name and the resource requested.

Previously, in our Renderer class, we passed the URL /myresources/myScript.js as an argument to the ViewHandler's getResourceURL() method. In the code sample below, we amend this to use the Weblet protocol instead.

```
ViewHandler handler = context.getApplication().getViewHandler();
String resourceURL =
       handler.getResourceURL(context,
              "weblet://org.myapp.html/myScript.js");
out.startElement("script", null);
out.writeAttribute("type", "text/javascript", null);
out.writeAttribute("src", resourceURL, null);
out.endElement("script");
```

The Weblet protocol-like syntax is convenient and easy to understand. The syntax starts with weblet://, followed by the Weblet name, e.g., org.myapp.html, and finally the path info or resource file, e.g., /myScript.js. Notice that neither the URL mapping nor the version number are included in the Weblet resource syntax. The Weblet URL mapping and version number are used by the WebletsViewHandler to create a resource URL that the Weblet will service.

When the component writer is not using Weblets, he would not be using the weblet:// resource path syntax and would distribute a separate installables zip. When the component writer moves to Weblets, he would start using weblet:// resource path syntax in the Renderer, and include the resources in the JAR. There is no benefit to using a mixture of these approaches for resources in the same version of the same component library.

Using Weblets in a JSF Application

In order to simplify setup for the application developer, component writers should select a default URL mapping for their component libraries. There is no need for the application developer to add any Weblet-specific configuration to the web.xml file, since the WebletsPhaseListener will be invoked automatically to service incoming requests for Weblet-managed resources.

Summary

As a new open source project, Weblets has tremendous possibilities to provide a defacto generic and configurable resource loading facility for Web clients and the JSF component community. The key differentiators are simplified packaging of JSF components and their resources, and a minimized overhead of installing and setting up JSF component libraries for a particular Web application project.

This article has explored a new way of packaging resources with JSF components. You should now be able to leverage Weblets in your own component library by including a suitable weblets-config.xml file and using the weblet:// protocol-style syntax to reference Weblet-managed resources.

In our next article in this series of building "Rich Internet Components with JavaServer Faces," we are going to look at how we can design JSF components using AJAX and Weblets.

Listing 1: Weblets configuration file, weblets-config.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<weblets-config xmlns="http://weblets.dev.java.net/config" >
  <weblet>
    <weblet-name>org.myapp.html</weblet-name>
    <weblet-class>
      net.java.dev.weblets.packaged.PackagedWeblet
    </weblet-class>
    <init-param>
      <param-name>package</param-name>
      <param-value>
        org.myapp.faces.renderer.html.resources
      </param-value>
    </init-param>
  </weblet>

  <weblet-mapping>
    <weblet-name>org.myapp.html</weblet-name>
    <url-pattern>/myresources/*</url-pattern>
  </weblet-mapping>
</weblets-config>
```

Listing 2: Weblets configuration file using 1.0 versioning for production

```
<?xml version="1.0" encoding="UTF-8" ?>
<weblets-config xmlns="http://weblets.dev.java.net/config" >
  <weblet>
    <weblet-name>org.myapp.html</weblet-name>
    <weblet-class>net.java.dev.weblets.packaged.PackagedWeblet</
          weblet-class>
    <weblet-version>1.0</weblet-version>
    <init-param>
      <param-name>package</param-name>
      <param-value>org.myapp.faces.renderer.html.resources</param-value>
    </init-param>
  </weblet>

  <weblet-mapping>
    <weblet-name>org.myapp.html</weblet-name>
    <url-pattern>/myapp/*</url-pattern>
  </weblet-mapping>
</weblets-config>
```

BY NOW THERE ISN'T A SOFTWARE DEVELOPER ON EARTH WHO ISN'T AWARE OF THE COLLECTION OF PROGRAMMING TECHNOLOGIES KNOWN AS AJAX!

REAL-WORLD
AJAX
ONE DAY SEMINAR

www.ajaxseminar.com

March 13, 2006

Marriott

Marriott Marquis Times Square
New York City

For more information
Call 201-802-3022 or
email events@sys-con.com

How, in concrete terms, can you take advantage in your own projects of this newly popular way of delivering online content to users without reloading an entire page?

How soon can you be monetizing AJAX?

This "Real-World AJAX" one-day seminar aims to answer these exact questions...

Led by "The Father of AJAX" himself, the charismatic Jesse James Garrett of Adaptive Path, "Real-World AJAX" has one overriding organizing principle: its aim is to make sure that delegates fortunate enough to secure themselves a place – before all seats are sold out – will leave the seminar with a sufficient grasp of Asynchronous JavaScript and XML to enable them to build their first AJAX application on their own when they get back to their offices.


Jesse James Garrett
Father of AJAX


Scott Dietzen
Creator of WebLogic, Ph.D., President and CTO, Zimbra


Bill Scott
AJAX Evangelist of Yahoo!


David Heinemeier Hansson
Creator of Ruby on Rails


Satish Dharmaraj
Father of JSP, Co-Founder & CEO, Zimbra


Rob Gonda
Best-Selling AJAX Author, CTO, iChameleon Group


Dion Hinchcliffe
Co-founder & CTO, Sphere of Influence Inc.


Ross Dargahi
Well-known AJAX Evangelist & Co-founder and VP of Engineering, Zimbra

Early Bird **$995**
(Before January 31, 2006)

Discounted Price **$1,195**
(Before February 28, 2006)

Seminar Price **$1,295**
(After February 28, 2006, and if any seat is available)

HURRY! LIMITED SEATING THIS SEMINAR WILL SELL-OUT CALL 201-802-3022 TO REGISTER!

Jeremy Geelan
Conference Chair, Real-World AJAX
jeremy@sys-con.com

MEDIA SPONSOR
ajax developer's journal

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

PRODUCED BY SYS-CON EVENTS

Joe Winchester
Desktop Java Editor

When Fixing Problems, Look Beyond Merely Improving the Existing Solutions

One way in which technology is adopted is when an existing process is automated and made more efficient, cheaper, or reliable. Another is when a technique or innovation is applied to an existing process to drastically alter the way it occurs. The disadvantage of the latter is that it requires the idea being sold to someone who has to change to adopt it, and thereby carries a risk of failure. Applying a technology to merely streamline an existing process is a simpler to adopt as the implementation merely involves oiling an existing solution.

Given the keystone that communication occupies in our lives, you would think that it would be an exemplary case of technology implementation. Ironically though this isn't what has occurred; instead it is one of the most recalcitrant disciplines.

An example is the telephone. When it was first rolled out, it was used by secretaries to read memos sent by company executives. The secretaries would write down the message and read it to the recipient's secretary who'd deliver and collect a reply before phoning it back. It was a streamlining of the previous process, which had involved the message being carried to a telegraph agent who would encode it in Morse and send it as a telegram. We take it for granted now that a phone message is a way of conducting a two-way conversation, but for the first telephones it took a long while before the message's author and recipient would talk to each other directly and cut out the additional latency and inefficiency introduced by having couriers in-between.

While with smug hindsight we can mock the early adoption of the telephone, I fear that generations to come will do likewise with our current usage of e-mail. It originates from a scenario where mail would be used to send memos within and letters between organizations. Once computers were added to the equation, instead of hand writing memos and letters, people were using word processors and sending printed output. With networking, the obvious implementation was to shortcut the process and simply send the mail electronically to the recipient. What has occurred though is a world where e-mail has almost become the means and the end itself. A colleague came by my desk a few days ago to ask if I had received the e-mail he had sent me, and then went back to his cubicle to await my reply. Our corporate phone system e-mails you when someone leaves a phone message. In some situations I have received e-mails from people asking for a convenient time to telephone. Leaving aside the politics, flame, and other idiosyncrasies that come to the fore with e-mail, it is surely a poor implementation of technology as a means to solving the general problem of increasing communication effectiveness.

There are other examples of where the path of least resistance has been used, when a technology rollout has been influenced heavily by the legacy it replaces, rather than the promise it offers. Numeric keypads are a case in point, where telephones lay out 123 on the top row, and computers 123 on the bottom. The computer arrangement descends from calculators and adding machines, so why is the telephone different ? It stems from an evolution that began with rotary phones with dials. The dial is turned clockwise to a fixed position and released, clicking on its return to generate pulses for each number; one for a 1, nine for a 9 and ten for a zero. The arrangement of the number was such that one was at the top, nine toward the bottom, and counter-clockwise next to it the zero. Nine and zero were infrequently used in phone numbers so they were the least easy to dial. When a numeric keypad for a phone was required, the engineers felt that, instead of just adopting the calculator key layout, keeping 1 at the top and 9 adjacent to 0 was easier for current phone dial users to migrate to. It was a decision based on following the path of least resistance to the adoption of technology, yet it has now created paradoxes such as the layout of soft keys on a telephony application being different from the physical layout of the keys on the computer's numeric keypad or the layout on a calculator application. It is a bizarre and seemingly insoluble legacy that stems from the initial solution being too closely aligned with the existing implementation rather than the new technology.

Apart from being interesting, these anecdotes provide lessons of failure where the common thread is that a technology is applied to an existing solution, rather than to the existing problem. When IT was first launched on corporations, buzzwords such as "business process re-engineering" were thrown about as jargon to embody the fact that computers would change the way the company worked, as well as merely improve its existing transactions. Instead, however, IT is now viewed as a cost center that has to purchase copies of operating systems and office applications, using whatever change remains to play with business innovation. It's akin to asking the sales department to pay a company's electric bills and catering costs before trying to generate new accounts, yet it passes off every day in boardrooms where IT managers have to fight daily for every dollar of their budget.

Whatever the future holds for IT, it has to come by looking at an existing scenario, and not merely attempting to tackle the old solution with a newer and faster technology, but peeling back the layers and working out what the original root problem was of the solution that's currently in place. The Internet has launched huge corporations that adopted technology as their means of doing point business rather than a bolt on, and the future is boundless as new problems are tackled with increasing bandwidth, mobility, and content type. I just hope that for each solution we aren't stuck with upside-down keyboards and implementations that hinder rather than aid communication, and that we all remember that the best solutions come from analyzing problems, not patching existing solutions.

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

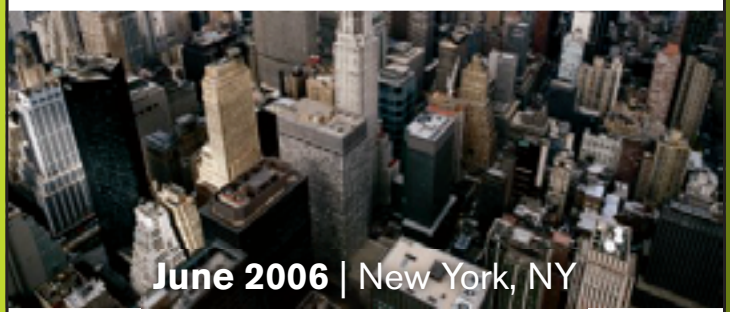
joewinchester@sys-con.com

ENGAGE AND EXPLORE...

The Technologies, Solutions and Applications that are Driving Today's Initiatives and Strategies...

CALL FOR PAPERS NOW OPEN!

SOA WebServices Edge 10th International conference+expo 06



The Sixth Annual SOA Web Services Edge 2006 East - International Web Services Conference & Expo, to be held June 2006, announces that its Call for Papers is now open. Topics include all aspects of Web services and Service-Oriented Architecture

Suggested topics...

- *Transitioning Successfully to SOA*
- *Federated Web services*
- *ebXML*
- *Orchestration*
- *Discovery*
- *The Business Case for SOA*
- *Interop & Standards*
- *Web Services Management*
- *Messaging Buses and SOA*
- *Enterprise Service Buses*
- *SOBAs (Service-Oriented Business Apps)*
- *Delivering ROI with SOA*
- *Java Web Services*
- *XML Web Services*
- *Security*
- *Professional Open Source*
- *Systems Integration*
- *Sarbanes-Oxley*
- *Grid Computing*
- *Business Process Management*
- *Web Services Choreography*

CALL FOR PAPERS NOW OPEN!

2006 ENTERPRISE> OPENSOURCE CONFERENCE+EXPO

The first annual Enterprise Open Source Conference & Expo announces that its Call for Papers is now open. Topics include all aspects of Open Source technology. The Enterprise Open Source Conference & Expo is a software development and management conference addressing the emerging technologies, tools and strategies surrounding the development of open source software. We invite you to submit a proposal to present in the following topics. Case studies, tools, best practices, development, security, deployment, performance, challenges, application management, strategies and integration.

Suggested topics...

- *Open Source Licenses*
- *Open Source & E-Mail*
- *Databases*
- *ROI Case Studies*
- *Open Source ERP & CRM*
- *Open-Source SIP*
- *Testing*
- *LAMP Technologies*
- *Open Source on the Desktop*
- *Open Source & Sarbanes-Oxley*
- *IP Management*

Submit Your Topic Today! events.sys-con.com

Sponsored by

WebServices JOURNAL · XML JOURNAL · .NET DEVELOPER'S JOURNAL · eclipse developer's journal · WebSphere JOURNAL · information STORAGE+SECURITY journal · wldj · JDJ · LINUXWORLD MAGAZINE · MX developer's journal · asp.netPRO · SD Times · CoDe · Software Test & Performance

*Call for Papers email: events@sys-con.com

Attention Exhibitors:

An Exhibit-Forum will display leading Web services and OpenSource products, services, and solutions

For Exhibit and Sponsorship Information - Call 201 802-3023

Produced by SYS-CON EVENTS

The Flexible Model

Loose coupling of Models in an MVC-based framework

by Man-ping Grace Chau

This article aims to illustrate how loose coupling of a Model in an MVC-based framework can be achieved by describing a real example – developing a framework for a Web-based XSD-XML generator, which is part of the Event Web research at the Infospheres Lab at Caltech. Why this is important is explained, along with a description of the various techniques used to accomplish the goal. Examples include: how a Model can be initiated in a modular manner; how to add dynamic properties to a Model without polluting the Model base classes; how to change the Model without affecting its existing operations; how the Model can be switched during runtime without affecting interactions with other components; how all these can be done if the Model is complex as in a DOM structure and is generated dynamically. Design patterns are largely employed to construct the framework; how the Eclipse XSD API should be used is also illustrated.

Why Only Controller and View?

When people discuss MVC they concentrate on View and Controller, asking "How can the View be changed easily to enhance user interface?" and "How can the Controller be easily changed to enhance the algorithm to handle user input?" Apache Struts, a Java Web application framework, is a very successful application of the MVC paradigm. Not surprising, Struts is also mainly concerned with View and Controller. They are linked flexibly – new Controllers can be added by sub-classing the Action class, and can be easily linked to Views as specified in a configuration file. In the same way, new Views can be easily linked to the old Controller by changing the configuration file. ActionForms are available to facilitate communication between the View and Controller.

Struts is successful because its framework is sufficient to handle most Web application development. Changes always take place in the Controller and View, e.g., the calculation of a discount in an online store frequently changes; the appearance of the Web application is often changed to enhance the user experience.

Why, then, would people ignore the Model? It's because the Model is typically static. For example, the customer and price model remains the same in most online shopping applications. So tight coupling between the Model and other components would normally not be a major problem during redevelopment. It's therefore reasonable for an MVC-based framework to put less emphasis on the Model. However, this isn't true in some cases, as shown in the XSD-XML Web generator below.

Model Matters

The XSD-XML Web generator here is part of the Event Web, developed in the Caltech Infospheres Lab headed by Professor K. Mani Chandy and Dr. Daniel M. Zimmerman. For more details, please refer to www.infospheres.caltech.edu. In this project, a Web portal is developed for users to interact with the Event Web and part of its functionality is to help users specify an XML file based on an XSD. In other words, an XSD-XML Web generator is needed with the flow chart shown below:

Only a simplified version of the generator is developed for the prototype. However, redevelopment is needed when:

1. More XSD features are supported to describe complex XML instances, e.g., annotations and attributes. The parsing process then becomes more complicated.
2. The XSD uses features (such as group and choices) that could generate XML instances with different structures. This means the users can choose their own DOM structure and dynamic UI is needed.
3. The type definition becomes more complicated. For example, the element content consists of an SQL statement to describe the price range "(Price < 10 AND Price >2) AND Price <> 5". In this case, an interactive tool is needed to help users specify the input and analysis on user input is needed.

Redevelopment– re-reading the code and modifying the program while preserving some of the operations – isn't pleasant. The framework exists to help programmers understand the program and know exactly what to change to make an enhancement without affecting the entire program. An MVC-based framework is a natural choice here and the major components can be easily identified as:

1. View – the input form
2. Model – the DOM structure generated by parsing the XSD
3. Controller – the link between View and Model to analyze user input and update the Model, responsible for forwarding a new View to the user

Man-ping Grace Chau is a student at the Chinese University of Hong Kong, majoring in information engineering.

mpchau2@ie.cuhk.edu.hk

Loose coupling of the Model plays a crucial role in this framework. The Model here is the DOM structure generated dynamically from the XSD file and redevelopment to support more complex XSD would give rise to the following design issues:

1. When more features of XSD are supported, the algorithm for generating the forms/files has to be changed. How can this be done without affecting the algorithm that manipulates the Model?
2. New operations like analyzing user input may be needed in complicated type definition. How can that be added without making the Model aware of it?

These two problems are related to how the Model is coupled with the other parts of the framework. In addition, our Model is generated dynamically by parsing an XSD. In complex events where users are allowed to define their own DOM structures, how can the Model be changed and presented by the Controller and View efficiently, given that the Model is not known until runtime?

The complexity of the XSD specification makes these problems worse. Manual effort is needed to determine how each XSD feature should be interpreted in making a form and generating the XML. For example, element declaration and type definition describes what the DOM structure should be like, facets of simple types specify how error checking should be done, and model groups specify the different forms of DOM structures that users can choose from. For the sake of completing the background information before discussing the framework design, we will briefly discuss XSD parsing here, which is how the Model is initiated. A tree structure shown in the following diagram is formed when the example XSD is parsed using Eclipse XSD API.

A common mistake is to assume that the tree structure generated by parsing the XSD is exactly the DOM structure of the target XML file, which is clearly not the case as shown. As mentioned, an XML instance specified by a XSD can have alternate DOM structures, so it doesn't make sense to expect that parsing the XSD will generate the corresponding DOM structure directly. Additional manipulation is needed to create the DOM tree of the target XML instance from the tree structure above. The corresponding DOM structure is shown below.

Shaping the Framework

One trivial way to structure the Model would be to build a DOM tree directly when parsing the XSD file. But such an approach has disadvantages:

1. DOM handling is both memory-hungry and computationally expensive, especially when users are allowed to re-structure the tree.
2. Some information such as the type of text node and possible alternate DOM structures can't be stored in DOM. Additional data structures are needed.
3. The processes of creating the DOM tree and parsing the XSD are tightly coupled, i.e., the process of parsing the XSD isn't made modular. When more XSD features have to be supported, the entire process can be affected. So can we fake a DOM tree to overcome these constraints?

Let's begin by building a family of Element objects that contain properties specifying the tag names, type information, etc. They go into ArrayLists and are linked together like a DOM tree as shown below. The Elements are classified by their types as specified in the XSD (e.g., simple type, complex type, etc.).

While building our DOM structure after the XSD is parsed, Elements of different types know the different forms of data they are looking for. For example, a built-in type element only looks for the tag name and the built-in type. A complex type element looks for the other elements that make up the complex type as well as the model groups that structure the nested elements. As a result, the parsing process becomes modular – each Element knows how to parse itself. When more advanced XSD features are needed, like group referencing in elements of complex types, only the parsing method of ComplexTypeElement has to be changed. How the other elements are parsed isn't affected. This technique can also be used to build an abstract syntax tree for a compiler or interpreter.

What's more, sub-classing can be done to support even more complicated elements (e.g., elements referencing substitution groups). Since they have the same interface for providing information, adding more Elements in the Model is transparent to the other components. For example, View calls the same "getTag" method on the Elements without knowing their concrete classes.

But this doesn't completely solve the problem of supporting extra XSD features. For example, sub-classing the Element won't support simple type elements with different facets, because 'simple type with facet' isn't classified as another type, and sub-classing like this would lead to an explosion of classes. Nor it is desirable to rewrite the entire SimpleTypeElement class to specify how the facet should be handled. In including attributes, the attributes should be added to the individual Element object instead of the entire class, since every Element doesn't need attributes, but this can only be determined at runtime when the XSD is parsed. To address this, a Decorator pattern can be used to attach

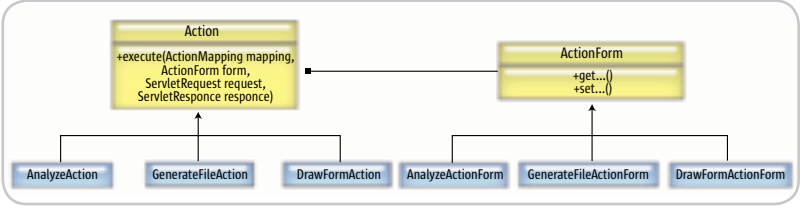


Figure 1 Action and ActionForm

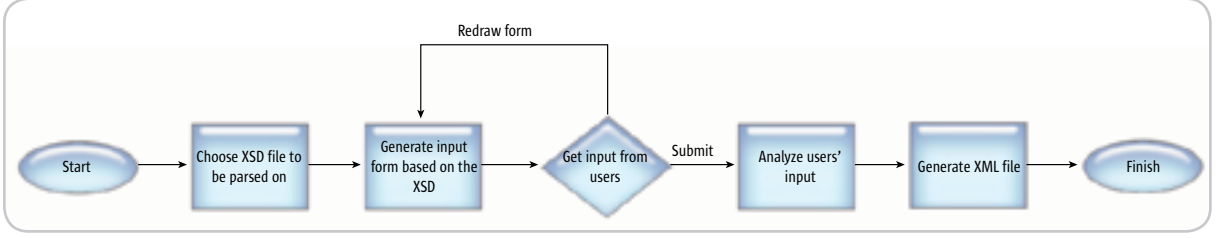


Figure 2 Workflow

additional responsibilities to an object dynamically without polluting the Element class:

1. Information about these additional features is stored in the Decorators.
2. The handling of these additional features (e.g., parsing) can be delegated to the Decorators.

A Decorator can even be composed with another Decorator to enhance its functionality according to the Composite pattern. For example, Attribute can own a Facet Decorator for specifying the attribute type.

To sum up, we have now described how our defined DOM structure can support different features of XSD for generating the input form and the XML file – a family of Elements is defined and different Decorators (additional features) contribute to different aspects of the Element. For example, Facet represents the validity of the node values. Of course it should be noted that the Element should only be a data keeper, except in the case of parsing, since only it knows how to initialize itself. How it's manipulated, (say represented in a form) is delegated to other parts of the framework.

Consequences

1. How different types of Element are handled (e.g., parsed) is made modular. Changing how one is manipulated doesn't affect another.
2. Adding Elements can be done easily by sub-classing the Element base class and they share the same interface to interact with the other parts of the framework.
3. Adding dynamic properties (e.g., attribute and facet) to Elements can be done by sub-classing the Decorator base class. They can be attached to the Elements while parsing. This prevents an explosion of Element sub-classes.
4. The handling of Decorators is made modular.
5. Decorators can enhance their functionalities dynamically by composing with other Decorators.

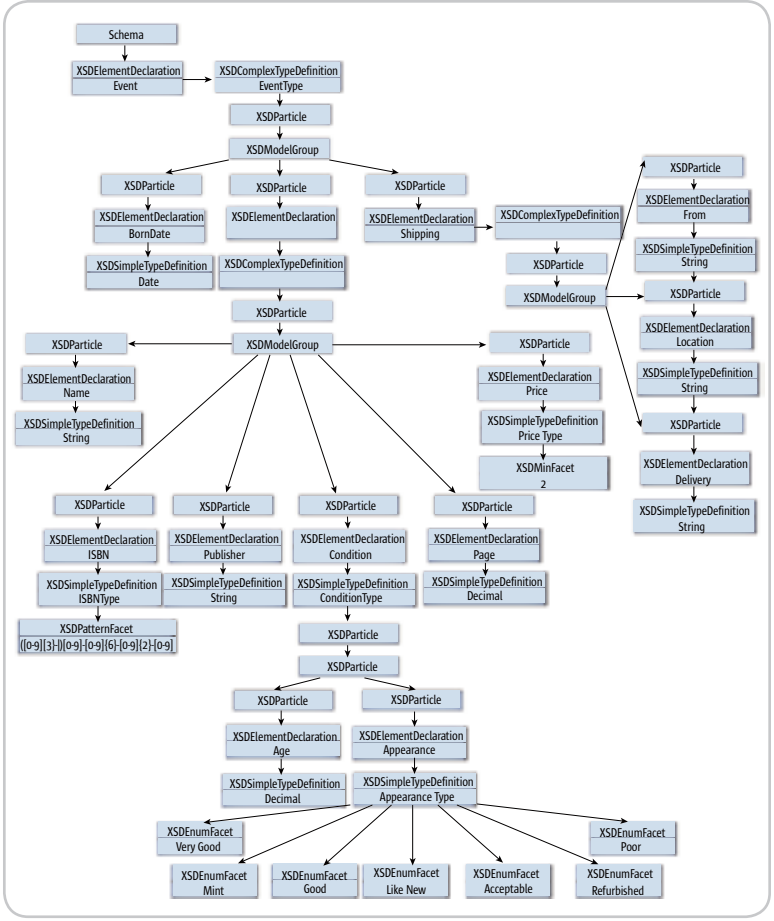


Figure 3 The schema returned

Hierarchy of Visitors

To support 'simple' XSD, only two operations are performed on the Model – print the Model as a form and generate a XML file based on the Model. A naïve implementation would let each Element implement a 'print' and 'generate' method to make it handle itself on request. However, this has three disadvantages:

1. When additional methods are needed (e.g., an analysis of user input), all Element classes have to be changed to support the additional method, which can be expensive.
2. Operations of the same nature (e.g., print form) are scattered across different Elements, which makes them hard to understand and manage.
3. The state generated by iterating through the Elements should be accumulated. However this can't be done unless the state is passed as an extra argument or stored as a global variable, which isn't easy to manage.

The Visitor pattern can be used to solve the problems. Related operations are packaged in the same Visitor object (e.g., PrintFormVisitor and GenerateFileVisitor). On request, the Element is asked to accept a Visitor to delegate the job (e.g., print itself). The Element also passes itself as the argument during the request, so the Visitor can carry out its operation based on the type of the Element. For visiting the built-in type element, the Visitor only asks for the tag and type; but for visiting a complex type element, the Visitor asks for the nested Elements that define the complex type and the model group. Double dispatch is possible here – the method invoked depends not only on the object called, but also on the caller object.

As a result, the Elements only act as data keepers in the framework and the operations for manipulating them are defined elsewhere – localized in Visitors, which are easier to manage. State can also be accumulated in Visitors when iterating through the Elements. This is how the different parts of the framework interact with Model – by owning these Visitors. For example, View owns PrintVisitor to draw the input form and Controller owns ValidateVisitor to do input validation.

Consequences

1. The addition of new operations for manipulating the Elements can be done by sub-classing the Visitor base class.
2. Similar operations on different Elements are localized in the same Visitor class. The manipulation of each Element

is specified by Visit methods with signature of their own types. Making changes to any of these has little to no effect on the others.

3. Elements and Decorators act as data keepers without providing any methods to handle themselves (except parsing).

Making It Work

To iterate through the self-defined DOM structure to print a form or generate a file, we need recursive functions. A simplified version of the recursive function that generates a XML file in pseudo code is:

```
public boolean GenerateNode(Node node, Document doc, Element parent)
{
      boolean isError = false;
      Element e = doc.createElement(node.getTag());
      parent.appendChild(e);
      if (node.getNestedElement() != null)
     {
               foreach (node.getNestedElement() as children)
               {
              isError = GenerateNode(children, doc, e);
                        if (isError)
                        {
                              return false; //return at once as
leaves have errors
                        }
               }
       else
      {
              //this is a leaf node, some more arbitrary operations
      }
        isError = validate(node); //arbitrary method to check valid-
ity of this node
        return isError;
}
```

So to make the Visitor iterate through the Model, the Visit method must be recursive. This can be done by asking the children to accept the Visitor (itself) again and pass in the necessary parameters.

```
isError = children.accept(this, doc, e);
// children would call visitor.visit(this, doc, e) to invoke recur-
sive call

Or this can be done by directly invoking the other visit method:

isError = this.visit(children, doc, e);
```

However, these methods break the Visitor pattern:

1. The first method: The accept method becomes dependent on the recursive operations – it has to conform to the return type of the recursive Visit method and includes the parameters needed for the recursive function.
2. The first and second method: All the Visitors can now have different signatures for the different recursive Visit methods.

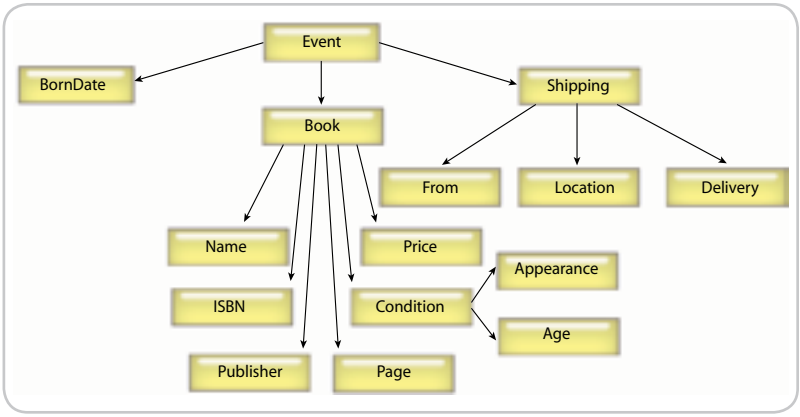


Figure 4 The self-generated DOM tree

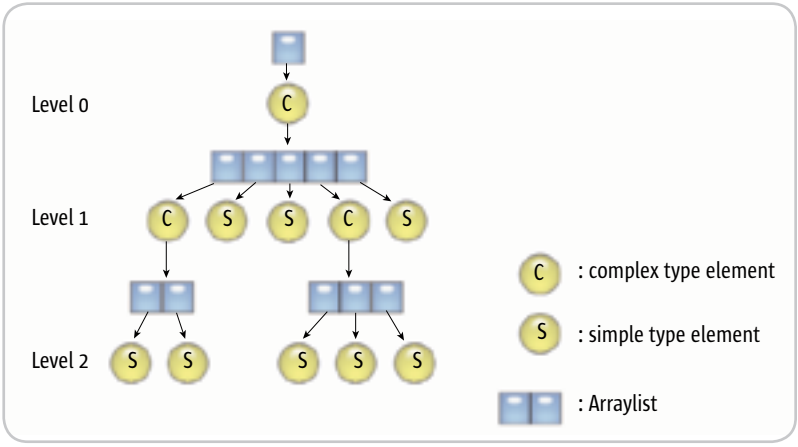


Figure 5 Our defined DOM structure

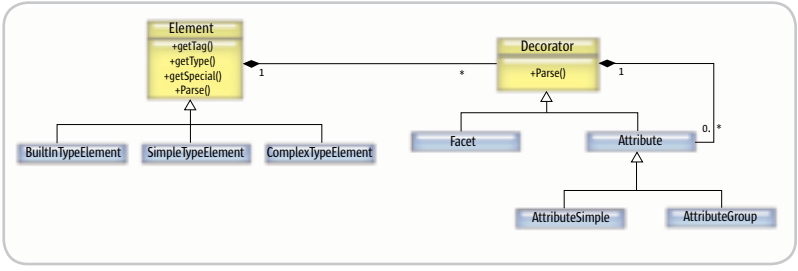


Figure 6 Elements and Decorators

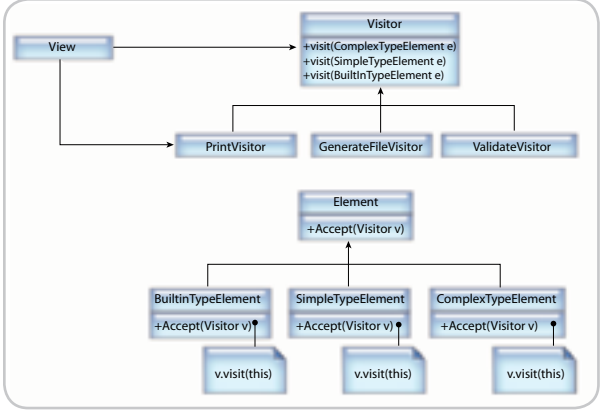


Figure 7 Visitor

To solve the problem, objects can be defined for keeping these parameters and return values in the Visitor. These object structures should be defined as private inner classes of the Visitor since they are solely used by the recursive visit method and no other object ever accesses them. How they are used is shown in Figure 8.

As a result, both visit and accept methods are void methods, don't take parameters that are specific to the recursive operations, and the Visitor pattern is preserved.

Consequences

1. The Visit method under the Visitor pattern can be made recursive to handle complex data structures.

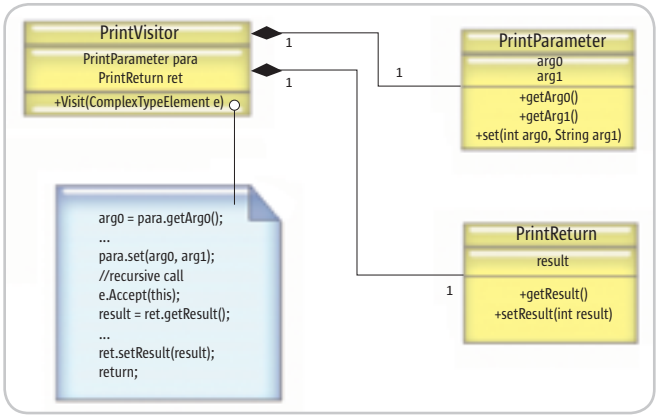


Figure 8 Recursive visit

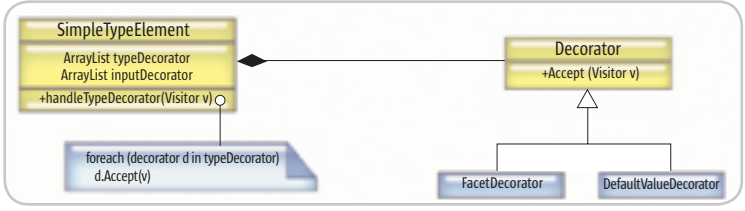


Figure 9 Strategy with Visitor

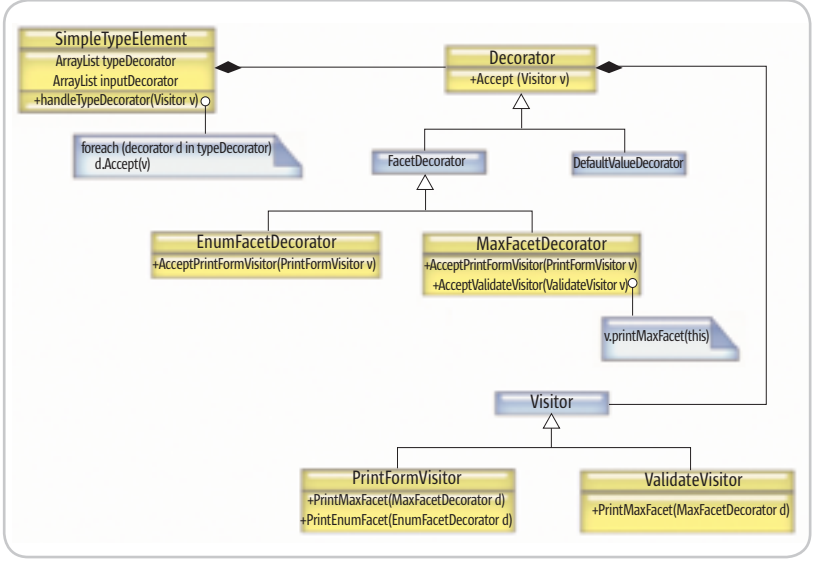


Figure 10 Visitor with reflection

Strategic Decorator and Reflective Accept

It's the Elements' responsibility to determine which Decorator should be used to service requests for different kinds of information. For example, when asked for a type restriction, the Element should forward the request to the Facet Decorator. It's desirable that when additional Decorators are added, the effort required to determine how the Decorators should be used is kept to a minimum and doesn't necessitate big changes to the Element class.

It's unavoidable that the parsing method of the Element has to be changed whenever new Decorators are added as when the Decorators are instantiated and attached dynamically to the Elements. This should be the state that determines how the Decorator is used.

Is it possible to make this the only part that requires changes when adding new Decorators? The answer is yes – by making use of the Strategy pattern

Let's begin by looking at how an Element maintains its set of Decorators.

During parsing, the Element instantiates the Decorators and puts them into different ArrayLists so that the Decorators delivering the same kind of information are grouped together. For example, different types of Facet Decorators are saved in the ArrayList typeDecorators. When the Elements are invoked by the Visitor to give certain information. For example, type. All the Decorators in the ArrayList typeDecorator are invoked to fulfill the request. According to the Strategy pattern, Decorators all share the same interface for providing the service, e.g., PrintInformation, so the client Element can make use of the encapsulated algorithm without knowing which Decorator it's using. When more Decorators are attached, only the parsing method of the Elements has to be changed to put the Decorator into the right ArrayList to identify its functionality. This technique can also be used in implementing a compiler/interpreter to store the user-defined type information. Instead of storing the Decorators in an ArrayList, a HashMap can be used.

This solution again scattered the operations of similar type into different Decorators, which is undesirable. To overcome this problem, the Visitor pattern can be applied again. Instead of asking the Decorators to perform the operations themselves like print information, they should again delegate the job back to the Visitor and act only as data keepers. A reference to the Visitor can be easily obtained. The Visitor passes itself to the Element when invoking data from Element (e.g., type information for drawing the form for PrintVisitor), so the Element can pass it to all Decorators in the ArrayList through the common accept method.

This isn't the end of the story. Some adjustments still have to be made. In the traditional Visitor pattern, there are multiple Visit methods with different signatures for all Decorators, and changes to the Visitor base class are needed whenever new Decorators are added. This may be feasible in handling Elements, but definitely not for Decorators. Not all Visitors need information from all Decorators. For example, the Decorator for specifying the default value is useful to the Visitor that prints the input form, but not to the Visitor that

validates input (assuming that users can't change the default value in the form).

Reflection is how to make the Visitor pattern more flexible. By using reflection, Decorators will be aware of the Visitor sub-class type during accept so that they can invoke the specific methods of the Visitor sub-classes to handle themselves. As a result, the Visitors no longer have to share the same interface in handling the Decorators. When new Decorators are added, only those specific Visitors that handle the new Decorators have to be changed by adding a method that indicates how the Decorator is handled.

On the other hand, when more Visitors are added, not all Decorator classes have to be modified. Their interfaces to communicate with the Visitors, namely the 'gateway' accept method, remain unchanged. Only those Decorators that are going to be manipulated by the new Visitor need an additional method to accept the new Visitor, which is invoked by refection in the 'gateway' accept method. As a result, Decorators are aware of the Visitor that they are sending requests to and know exactly which methods of different Visitors have to be invoked to handle themselves. The overall idea is presented in Figure 10.

The Accept method of Decorators is shown below:

```
public boolean Accept(Visitor v)
{
 String methodName = v.getClass().getName();
 //method name starts from Accept and then the class name of visitor
 methodName = "Accept"+ methodName.substring(methodName.lastIndexOf('.')+1);
 try
 {
  Method m = getClass().getMethod(methodName, new Class[] { v.getClass() });
  m.invoke(this, new Object[] {v});
 }
 catch (Exception e)
 {
  System.out.println(e.getMessage());
  return true;
  //indicate error
 }
 return false;
}
```

This can be applied to other cases where both the Model and the operations on the Model need constant changes. What's more, programmers would have an explicit understanding of how the Decorators are handled differently by different Visitors. At the same time, the Decorators still act only as data keepers, their manipulation is delegated to Visitors, and operations of similar kinds can be localized in the same Visitor.

In summary, the following must be done when a new Decorator is added:

1. Determine the type of information revealed by this Decorator and set the properties
2. Implement the methods for handling this Decorator in different Visitors
3. Let the Decorator invoke those methods of Visitor in the specific Accept'visitor' methods

Consequences

1. When adding a new Visitor, only those Decorators that are going to be manipulated by the new Visitor have to provide an accept method for that new Visitor.
2. When adding a new Decorator, only those Visitors that are going to manipulate the Decorator have to provide a method that's meant to be invoked by the new Decorator to handle themselves.
3. By using the Strategy pattern, the Element can forward the requests to the appropriate Decorators without acknowledging which Decorator it's handling. When adding new Decorators, only the parsing part of the Element has to be changed.

Chain of Visitors

While generating the XML file from the Model, it's desirable to validate at the same time but walking through the same Model twice would be expensive. However, these are completely unrelated operations. How should they be combined while maintaining their loose coupling? The answer is by using the Chain of Responsibility pattern. In validating while generating a file, the GenerateFileVisitor can make the Element accept a ValidateVisitor during the generation process. When there are errors, the GenerateFileVisitor will continue to iterate through the remaining Elements to finish the validation process without generating the file. As a result, the ValidateVisitor can rely on the GenerateFileVIsitor to iterate through the Elements and the algorithm to iterate the same Model need not be written twice.

The good thing about Chain of Responsibility is that operations can be extended without affecting the previous operations. For example, when an analysis of user input is needed, the ValidateVisitor can make the Element accept an AnalysisVisitor to change the user input to some other form (e.g., an SQL statement) while generating the file, without changing anything in the GenerateFileVisitor. Since all the Visitors share the same interface, the Visitors can even be switched while iterating through the Elements without making the Elements aware of it.

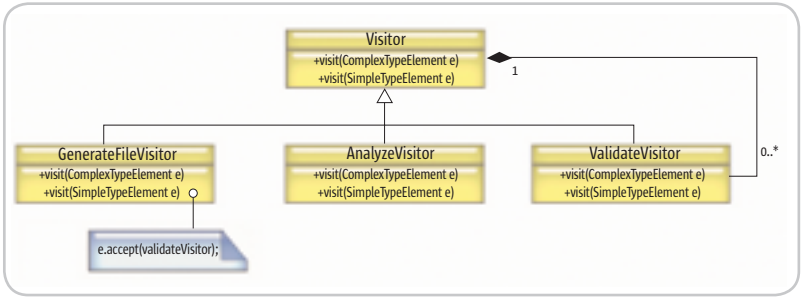


Figure 11 Composite of Visitors

Consequences

1. Manipulation of the Elements is delegated to the Visitors and the operations can be extended by making the Elements accept nested Visitors without affecting what the current Visitor is doing.
2. The Visitor can be switched while iterating through the Elements without the Elements being aware of it. State accumulation can be maintained by passing self-defined data structure as parameters.

Summary

The MVC-based framework presented here emphasizes the loose coupling of the Model with the other components. This is important if the application often requires the Model and the algorithm for handling the Model to be updated. The following Design Patterns are employed to achieve this goal:

1. ***Decorator*** – Adding dynamic properties to the Model without polluting the Model classes.
2. ***Composite*** – Models can be composed from multiple Models to enhance functionality.
3. ***Visitor with reflection*** – Operations on the Model can be added without changing the Model base class; changes in the Model don't affect the Visitor base class
4. ***Strategy*** – Other parts of the framework can interact with the Model without being aware of the concrete classes they're dealing with.
5. ***Chain of Responsibility*** – The functionalities of operations on the Model can be extended without affecting the previous operations.

A way to implement recursive a visit method was also introduced to help the Visitor iterate through complex data structures.

Acknowledgments

I would like to thank my mentors Professor K. Mani Chandy, Dr. Daniel M. Zimmerman, and Mr. Jonathan Lurie Carmona for their unfailing guidance and support. Thanks to team members Mr. Weilin Shao, Mr. Mehran Shahir, and Mr. Zachary Henson for interesting discussions. The Event Web project is supported in part by the NSF Grant CCR-0312778 named "Information Infrastructures for Crisis Management."

Resources

- Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional Computing Series.
- K. Mani Chandy, Brian Emre Aydemir, Elliott Michael Karpilovsky, and Daniel M. Zimmerman. *Event Webs for Crisis Management*, IASTED International Conference on Communications, Internet and Information Technology, 2003.
- Jeremy Blosser, Reflect on the Visitor Design Pattern, http://www.cs.joensuu.fi/pages/ageenko/teaching/OOD/DD.pdf

JAVA DEVELOPER'S JOURNAL Advertiser Index

Advertiser	URL	Phone	Page
AJAX Seminar	www.ajaxseminar.com	201-802-3022	49
Altova	www.altova.com	978-816-1600	4, 19
Arcturus Technologies	www.arcturustech.com	703-822-4582	31
Azul Systems	www.azulsystems.com/developerfreedom	650-230-6691	29
ceTe Software	www.dynamicpdf.com	800-631-5006	43
InterSystems	www.intersystems.com/cache4p	617-621-0600	7
IT Solutions Guide	www.itsolutions.sys-con.com	888-303-5282	59
Java Developer's Journal	www.jdj.sys-con.com	888-303-5282	45
Jinfonet Software	www.jinfonet.com/jp12	301-838-5560	37
MapInfo	www.mapinfo.com/sdk	800-268-3282	21
Northwoods Software Corp.	www.nwoods.com/go	800-434-9820	41
Parasoft Corporation	www.parasoft.com/jdjmagazine	888-305-0041	Cover I
Perforce	www.perforce.com	510-864-7400	Cover II
Smart Data Processing, Inc.	www.weekendwithexperts.com	732-598-4027	61
Software FX	www.softwarefx.com	800-392-4278	Cover III
SYS-CON Events	www2.sys-con.com/events	201-802-3023	51
SYS-CON Newsletters	www.sys-con.com	888-303-5282	57
Synaptris	www.intelliview.com	866-99IVIEW	17
WebAppCabaret	www.ngasi.com/jdj.jsp	866-256-7973	23
Windward Studios, Inc.	www.windwardreports.com	303-499-2544	9
Xenos	www.xenos.com/VAN	888-242-0695	35
Xythos Software, Inc.	www.xythos.com	888-4XYTHOS	13

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

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

–continued from page 3

4. 2006 will be the year of acceptance of the importance of roles in the world of identity management and provisioning. Bridgestream is the leader in role management integrating with the leaders in identity management, directory services, and provisioning.
5. The two trends that will not be new for 2006 but that will continue their growth are Software as a Service (SaaS) or on-demand software and open source, which continues to find acceptance in the enterprise.

Our next set of predictions comes from **Jim Milbery**, CTO for Chicago Growth Partners in Chicago with 30-plus companies under his wing (.NET, Java, ColdFusion, Python – "you name it," as he says). He also acts as the "virtual CTO" for a number of companies in his portfolio.

JIM MILBERY
SANs, AJAX, Web 2.0, Blog consolidation, InfoSec

1. *Data storage:* The proliferation of blogs and the raw size of XML documents (and everything is XML these days) are going to drive us to a new emphasis on storage (SANs in particular).
2. *AJAX everywhere:* IE gets new life out of the proliferation of AJAX. More high-profile sites are going to adopt AJAX as a means of extending the life of the browser in the near term. We may even see the return of some application-development tools around AJAX (something more than just component libraries).
3. *Dashboard apps:* Even with the proliferation of AJAX we are going to see a serious rise in client-specific apps that are based on Web 2.0 technologies – think iTunes.
4. *Blogging acid-reflux:* The massive interest in blogging continues to rise, but reliance and confidence in individual blogs sags; high-profile blogs that are industry-specific begin to dominate and provide a bit of "editing" to the process.
5. *William Strunk Jr. rolls over in his grave:* The illustrious author of The Elements of Style officially rolls over in his grave. I thought that basic writing skills were bad as seen in e-mail documents, but blogging takes things to a whole new level of poor grammar and punctuation.
6. *Information security:* We start to get serious about protecting applications during the coding process, not just as an afterthought.

Next up is **Alan Williamson**, technology evangelist for SpikeSource and distinguished former editor-in-chief of *JDJ*, as well as chief architect of BlueDragon.

ALAN WILLIAMSON
Java, BitTorrent, Googlecrash, Adobe, IE

Here are my modest predictions for 2006:

1. Java has been in the dark for the past few years; its time to come back around again is here. Sun has some interesting initiatives in the pipeline.
2. The movie industry will wake up to BitTorrent (and the likes) and actually figure out a way to utilize this revolution instead of trying to close it down. You can't push back the tide. The BBC is going to be launching BBC2 as the first broadband television channel in 2006.
3. Google shares fall or even crash. Everything that goes up has to come down and, contrary to popular belief, they aren't the biggest player on the Internet and people will start distrusting them as Microsoft and Yahoo! crank up their offerings.
4. In fear of Microsoft Vista (and AJAX), Adobe will offer all Flash development tools for free, which will result in a major surge in adoption.
5. IE7 will probably more than likely eclipse Firefox again.

J.P. Morgenthal, managing partner for the IT consultancy Avorcor and the author of *Enterprise Information Integration: A Pragmatic Approach*, is as usual very forthright in his foresight.

J.P. MORGENTHAL
VPMNs, AJAX, VoIP Phones, SaaS, Semantic Technologies

1. *Private mail networks:* With people getting slammed I believe we will see the rise of VPMN (Virtual Private Mail Networks). Essentially, these are analogous to VPNs, allowing private network traffic to run over the public backbone. They use common SMTP protocols to deliver mail, but unless you have permission to send mail to the recipient the mail will be rejected.
2. *AJAX:* We will see the rise of even stronger support for more powerful portable client-based applications based on Web protocols.
3. *Composite applications:* With the rise of SOA and BPM, it's going to get even easier to develop applications that require less low-level coding skills and that are more flexible and can respond faster to changes in business.
4. *VoIP phones:* Advancements and growth in high bandwidth wireless networking means that wireless devices will be IP addressable, which means that the next wave of phones will leverage the public Internet for phone communications and common WAN/LAN. Windows CE and Palm devices will be able to provide voice services. Gone are the days of buying a phone dedicated to a single network provider.
5. *Self-publishing:* Garth Brooks and Wal-Mart, LuLu, MusikMafia. These names all represent a rise in successful self-publishing. Books, magazines, and music are all media that are being self-published over the Internet. Soon, this will be expanding to software as Software as a Service (SaaS) becomes more popular.
6. *Metadata:* Metadata is finally being recognized as a critical enterprise asset. It's now being managed properly and leveraged for its properties for automation.
7. *Semantic technologies:* People and organizations are finally starting to see the value in being able to describe data in context and defining the relationships between data. Semantic technologies enhance and extend the basic power realized by relational database technologies to data anywhere in the world.

JDJ's enterprise editor, **Yakov Fain**, has 10 predictions, several of them directly involving Java.

YAKOV FAIN
Java 5.1, AJAX, "CSMB," Outsourcing, Yahoo!

1. Enterprises will finally start using Java 5. The sooner the 5.1 version is released the better.

2. AJAX hype will calm down. AJAX is an interesting technology and will become one of many techniques used in Web application development. Nothing more.
3. Fat clients will be more widely used in distributed enterprise applications. Java still has a chance to be used in this area, if someone will create an IDE with an easy-to-use and powerful Swing GUI designer. JDeveloper and NetBeans are leading here. Adobe (formerly Macromedia) tools will become more and more popular.
4. Smart development managers will start creating mixed open source/commercial environments. For example, you can use open source J2EE servers in Dev and QA and their commercial counterparts in Prod and Contingency environments. The same is applicable to DBMS, messaging, et al. Some open source vendors are already moving in this direction by creating products that are 100% compatible with particular commercial tools.
5. A new software architecture for small and mid-size businesses should arise. IMHO a good candidate is what I call "Client/Server Message Bus" (CSMB): a set of client/server applications can talk to each other using open source messaging and an enterprise service bus. Note: Client/server applications can have more than two tiers, e.g., RMI client, RMI Server and DBMS.
6. Programming will become the trade of the younger generation. Middle-age programmers will be leaving the coding arena and moving to business analysis and management. You can't beat a 25-year-old Indian programmer who's ready to join any project tomorrow (in any place on Earth), sharing a room in so-called guest apartment. The code quality of such a programmer may not be as good as was expected by the employer, but this will be a little secret for some time, and smart kids will have enough time to learn how to program on the job.
7. A number of CIOs will come out of the closet and publicly admit that the real cost of outsourced projects is high, because for every two young Indian programmers, you need a local business analyst who will write super-detailed functional specifications and validate their work. But outsourcing is here to stay (at least in the U.S.) and not because overseas programmers charge less, but because just finding local programmers will become a difficult task.
8. Yahoo! will come up with some new innovative Web products that will be able to compete with Google's software. If not Yahoo!, who else?
9. By the end of the year the broadband Internet will give DSL and cable Internet a run for its money. The wireless companies just need to cut the prices of their broadband service, and the masses will start leaving their "traditional" ISPs.
10. Java use will steadily increase despite the fact that various replacements are being offered. Java is more than an excellent object-oriented language enriched by tons of productivity libraries (networking, multi-threading, security, etc.). It's a mature and proven platform for development of all kinds of applications for all kinds of hardware. Java in programming plays the same role as English in the real world: no one says that the Italian language will replace English any time soon; on the other hand, songs in Italian sound great.

Erik C. Thauvin, as befits the author of Erik's Linkblog and owner of Thauvin.net, ranged far and wide in his predictions. They started with combative opinions on RoR and Web 2.0.

ERIK C. THAUVIN

RoR, Web2.0, Open Source Java, IE 7, Google, Yahoo!, spam, VoIP, and WiFi

1. Ruby (on Rails) and such will still be touted as taking over Java, but in reality will be as insignificant as they are today.
2. Web 2.0 will solidify its status as a powerful buzzword. A lot of fluff, very little stuff.
3. Sun will once again dangle the open source carrot as Mustang gets closer to its release date.
4. The IE 7 rate of adoption will be phenomenal, especially compared to Firefox.
5. Sixty percent of Google's services will still be in "beta."
6. Yahoo! will be the first Internet portal to come up with a compelling set of mobile-based services.
7. No spam salvation. Many will try, all will fail.
8. VoIP and Wi-Fi will become even more synonymous.

So, let's have your own contributions: e-mail them please to 2006predictions@sys-con.com.

Acknowledgments

Parts of this article were informed by discussions with SYS-CON editors, writers and columnists, including Sean Rhody, Israel Hilerio, Bill Ray, Mark Hinkle, Rob Gonda, and Dion Hinchliffe.

Looking Back, Looking Ahead

Onno Kluyt

2005 may be remembered as the year of eating cake. I had the amusing honor of singing "Happy Birthday" and eat cake in Sao Paolo, Ede, San Francisco, Tokyo, and a couple more places as many Java groups and organizations around the world wanted to be part of the 10th year of Java technology. Together we journeyed from being amazed at a dancing Duke in a Web page to the full-frame, full-speed 3D graphics in today's computer games, all with the same technology. Just this year many achievements and events took place that deserve a mention, and I'd like to share my thoughts with you of what can be behind some this and what they may mean for the year ahead.

Let's start with the JCP Membership, which continued to increase. This year, as in previous years, the community increased by about 12–15%. Perhaps not too surprising. What pleased me is that the Java Community has become more inclusive of industries and geographies around the world, as illustrated by SouJava's joining from Brasil (the world's largest JUG and the first JUG to join), Tata Elxsi from India (worldwide provider of customized design solutions), and ChinaMobile from China (the first mobile service provider from that country to join).

An effort I lead at Sun in support of the goals of the JCP is Sun's TCK Scholarship Program. Established in 2002, the scholarship helps ensure that the cost of the rigorous compatibility testing process is not a barrier for not-for-profit organizations, universities, or qualified individual developers who want to build compatible implementations of JSRs led by Sun. Since this scholarship was established, quite a number of projects have benefited from it. In August, Sun extended the program for another three years. At http://java.sun.com/scholarship/ you'll find a list of scholarship recipients and the set of qualifications applicants are required to meet. You can send questions and applications to the Review Board at tck-review-board@sun.com. One scholarship application I would not at all be surprised to receive in the new year would be one from Project Harmony by the Apache Software Foundation for the Java SE 5 technology test suite.

Speaking of open source software, 2005 saw a further proliferation of this methodology and lifestyle. Apache's Harmony project is an example. Sun played its own part by moving its implementations of Java EE 5 and JSR 208, Java Business Integration, under the CDDL license. At java.net you can directly participate in the development of the next versions of the Java SE and Java EE platforms. While the Java SE 5 and Java SE 6 projects (aka JDK Community) haven't progressed that far using an open source software license,

they show that not only did open source software as a licensing model continue its march in 2005, but that the lifestyle it represents – the desire for openness and transparency – also continued its march. This will clearly carry forward in 2006. Within the JCP all this plays too. Compared with a few years ago, participants in the community and observers from the side will now expect (and often demand) that the specification development process itself also be more visible, more tangible, and have higher interactivity. The JCP took strides toward this trend when in JCP 2.6 it made basic rule changes to make all spec reviews public and to make an Expert Group's work transparent through observers and other means. My team, the JCP Program Office, will be focusing on two things the coming year that speaks to this. We will encourage and entice Spec Leads and Expert Groups to be as transparent, open, participative, and inclusive as they have the courage to be. And we will be rolling out substantive changes to the JCP.org site, providing broad sets of tools to Spec Leads and Expert Groups so that they can easily develop specifications collaboratively and interactively with the community at large.

Spec Leads have one of the hardest jobs in the Java community: to bring together a group of developers, from differing walks of life and with often competing goals, to develop high-quality Java technology specifications that will enjoy wide adoption, and do all that on predictable and reasonable schedules. Compared with the first three years of the JCP, JSRs now complete 100–120 days faster on average. To recognize the hard work and expertise (both technically and often also psychological…) of these developers, the JCP started the Star Spec Lead program in 2005. We will continue this in the new year. The goal of this program is not just to give these individuals well-deserved attention but also to enable the transference of their expertise to fellow Spec Leads and community participants in order to continue to raise the knowledge-level and performance of the Community.

This year saw the return of the first Java technology to enter the Java Community Process. The real-time specification for Java was the effort that started the JCP in December 1998. Now, this group of industry experts have started JSR 282 to develop the next version of this technology, which should complete during the course of 2006. You should be able to look forward to many great JSRs to finish in 2006: Java EE 5 (JSR 244), Java SE 6 (JSR 270), Mobile Architecture for CLDC (JSR 248), Mobile Architecture for CDC (JSR 249), and MIDP v3 (JSR 271), among many others.

Stay tuned to jcp.org for more cool JCP program activities and opportunities to participate actively in Java technology standards innovation. If you're not a JCP member already, consider becoming one as one of your New Year's resolutions!

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

onno@jcp.org

SoftwareFX
Bringing Your Data, Business & Strategy Into Focus.

New! version 6.2
Now Includes Maps!

All New Flavors...
Same Great Taste of Java!

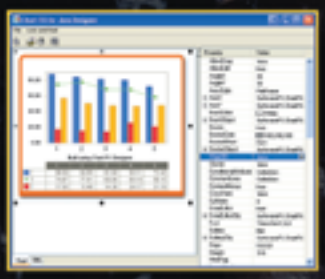


Chart FX for Java Designer

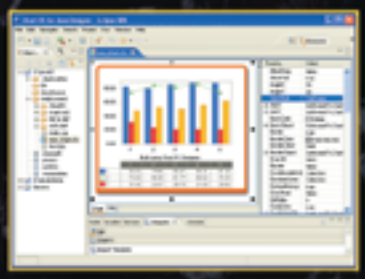


Chart FX for Java - Eclipse

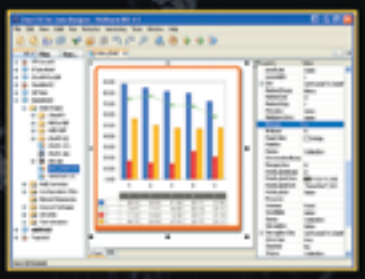


Chart FX for Java - NetBeans

Chart FX for Java 6.2

Now offers seamless integration within Eclipse and NetBeans, Chart FX for Java 6.2 brings New features include Smart & dynamic axis labeling, conditional marker and axis attributes, extended URL linking, multiple chart panes and robust annotation objects. One of the most dynamic new features is the inclusion of the Chart FX Maps Extension as a permanent fixture within Chart FX for Java. ➤

Chart FX

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

The Developer Paradox:

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

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

- **Automated:**
  Automatically analyzes your code, applying over 500+ industry standard Java coding best practices that identify code constructs affecting performance, reliability and security.

  Automatically generates and executes JUnit test cases, exposing unexpected exceptions, boundary condition errors and memory leaks and evaluating your code's behavior.

  Groundbreaking test case "sniffer" automatically generates functional unit test cases by monitoring a running application and creating a full suite of test cases that serve as a "functional snapshot" against which new code changes can be tested.

- **Extendable:**
  Industry standard JUnit test case output make test cases portable, extendable and reusable.

  Graphical test case and object editors allow test cases to be easily extended to increase coverage or create custom test cases for verification of specific functionality.

- **Integrated:**
  Integrates seamlessly into development IDE's to support "test as you go" development, and ties into source control and build processes for full team development support.

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

PARASOFT®
We make software work.™

Automated Software Error Prevention™