

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

# JDJ


JDJ.SYS-CON.COM

VOL.11 ISSUE:2

**Coming...**  
**March 13, 2006**  
**New York City**  
[www.ajaxseminar.com](http://www.ajaxseminar.com)  
**Marriott**  
 Marriott Marquis  
 Times Square, NYC

**SEE PAGE 51 FOR DETAILS**

REAL-WORLD  
**AJAX**  
 ONE DAY SEMINAR



# WHAT IS POJO PROGRAMMING?

*Introducing POJO application development*

## PLUS...

► Moving to SOA  
in J2EE 1.4

► Seam: The Next Step in the  
Evolution of Web Applications

► Spring and EJB 3.0  
in Harmony

RETAILERS PLEASE DISPLAY  
UNTIL APRIL 30, 2006

\$5.99US \$6.99CAN

03>



0 09281 01751 6



# Java 911

## Put out code fires with JProbe®

Tired of constantly fighting fires? Extinguish code issues and prevent future memory flare-ups with Quest Software's JProbe.

Discover the root cause of performance and memory issues in your Java applications. Easily test and fix your applications without any code changes. And even pinpoint performance problems down to the line of code. All with award-winning JProbe.

Stop the fire drills. Gain confidence in your code with JProbe.

---

Download a trial of JProbe at:

**[www.quest.com/911](http://www.quest.com/911)**

---



## 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: **Bahadır Karuv, PhD**

## Writers in This Issue

Yakov Fain, Jeremy Geelan, Andrei Iltchenko, Ales Justin, Boris Minkin, Norman Richards, Chris Richardson, Pankaj Tandon, Joe Winchester, Michael Yuan

To submit a proposal for an article, go to  
<http://jdi.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](mailto: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.

## ©Copyright

Copyright © 2006 by SYS-CON Publications, Inc. All rights reserved. No part of this publication may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopy or any information storage and retrieval system, without written permission. For promotional reprints, contact reprint coordinator Megan Mussa, [megan@sys-con.com](mailto:megan@sys-con.com). SYS-CON Media and SYS-CON Publications, Inc., reserve the right to revise, republish and authorize its readers to use the articles submitted for publication.

Worldwide Newsstand Distribution  
 Curtis Circulation Company, New Milford, NJ

For List Rental Information:

Kevin Collopy: 845 731-2684, [kevin.collopy@edithroman.com](mailto:kevin.collopy@edithroman.com)  
 Frank Cipolla: 845 731-3832, [frank.cipolla@epostdirect.com](mailto:frank.cipolla@epostdirect.com)

Newsstand Distribution Consultant  
 Brian J. Gregory/Gregory Associates/W.R.D.S.  
 732 607-9941, [BJGAssociates@cs.com](mailto:BJGAssociates@cs.com)

Java and Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc., in the United States and other countries. SYS-CON Publications, Inc., is independent of Sun Microsystems, Inc. All brand and product names used on these pages are trade names, service marks or trademarks of their respective companies.



FROM THE GROUP PUBLISHER

# As Oracle Recommits to Java, Sun Sweeps 2005 RCAs



Jeremy Geelan



Every year for the past 10 years, SYS-CON Media's "Readers' Choice Awards" have given the multiple constituencies we serve – developers, architects, IT managers, vendors – a chance to exercise their democratic rights, not just through the ballot box but also through the nomination process. The products, tools, and services voted on in any particular set of awards are also nominated from within the community. If there are sins, either of omission or commission, then it is accordingly to the community itself that one needs to look, not to *JDJ* or *SOA Web Services Journal* or *LinuxWorld Magazine* or any of the other SYS-CON publications that hold RCAs each year.

Looking at the Java results, in a year in which the *JDJ* Readers' Choice Awards were given to winners and finalists in 26 distinct product categories, one of the most striking things was the strength of Eclipse. And the simultaneous – defiant, almost – resurgence of Sun.

In spite of its name, in other words, the Eclipse IDE doesn't mean that "SUNset" is on its way any time soon. Just consider the sidebar below with the Sun results gathered into one place. With an extremely low barrier to entry, as in, free, it would seem that awareness of Sun's software portfolio is growing fast. This was a point picked up on right away in the Java blogosphere.

"I think what what SYS-CON is doing is good for the industry," blogged John Klingan, the day the results hit the news wires. "Popularity (awareness) is important for a product's success. Sometimes popular products are the best, sometimes not. Just take the results with a grain of salt. The best way to find out is to try it yourself. Easy to do when it is *free*." [My emphasis.] Klingan actually works for Sun, albeit not in Corporate, but his point is no less valid for that.

Another Sun blogger, a tad better known – not only within Sun but industry-wise – is president and COO Jonathan Schwartz, arguably the highest-profile blogger anywhere in the Valley. The power of his corporate blog was driven home this month when he used it to set the record straight after some idle speculation that there was some sinister significance to his not appearing at the Sun-Oracle lovefest held in the Redwood

Shores, CA, headquarters of Oracle, in which Oracle committed to Java and the Java Community Process for the next decade.

Sun was represented by Scott McNealy at the so-called "Town Hall Meeting," in which the audience was drawn from the respective staffs of the two companies (plus invited press such as SYS-CON Media and SYS-CON TV), just as Oracle was represented by Larry Ellison. These are, after all, the two longest-serving CEOs in the industry.

"Please don't read [anything] into my not being at Sun's recent announcement with Oracle," Schwartz wrote, adding: "There are better conspiracies."

For those who missed the press event, he then went on to explain: "Sun and Oracle just announced a broad-based reinvigoration of our working together – we announced a new 10-year partnership once again endorsing the Java Community Process, dousing any possible fear that the Java platform was at risk of fragmenting."

Sun also announced Oracle's adoption and endorsement of NetBeans, Schwartz added, "building on the groundswell of support we're seeing for NetBeans 5.0."

"We announced a special promotion for Oracle on Sun," he continued, "through which Oracle customers can now purchase Oracle's flagship database at 50% off by running on Sun. That alone is reenergizing our respective field organizations..."

Where in the world was Schwartz? In Mexico, it turns out, energizing Sun's field force there. In the next issue of *JDJ*, we are pleased to say, he will again be writing for us; his last article, "The End of Middleware," was one of the best-read Java articles anywhere in 2004.

## Sun's Results in 2005 Java Developer's Journal Readers' Choice Awards

**Best Web Services Platform**  
 Winner: Java EE

**Best Framework for SOA and Web Services**  
 Winner: Java Web Services Developer Pack

**Best SOA IDE**  
 Winner: NetBeans/Java Studio Enterprise

**Best XML Parser**  
 Winner: Java API for XML Processing

**Best XML Utility**  
 Winner: Java API for XML Processing

**Best SOA or XML Training**  
 Winner: Sun Java Web Services Developer Pack Tutorial

**Best SOA or XML Site**  
 Winner: <http://java.sun.com/webservices>

**Best SOA Security Solution**  
 Winner: JWSDP XML and Web Services Security

**Best SOA Portal Platform**  
 Winner: Sun Java System Portal Server

**Best SOA IDE**  
 Winner: Sun Java Studio Enterprise

**Best Java Training**  
 Winner: Java BluePrints

**Best Java Virtual Machine**  
 Winner: Java SE

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](mailto:jeremy@sys-con.com)

# 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](http://www.altova.com)

Also available in the Altova XML Suite.



# JDJ contents

## JDJ Cover Story

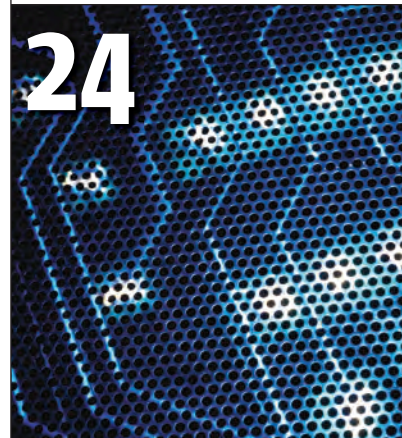
**32**

# WHAT IS POJO PROGRAMMING?

by Chris Richardson *Introducing POJO application development*

## Features

**24**



### Seam: The Next Step in the Evolution of Web Applications

by Norman Richards

**54**



### Developing Web Services Eclipse Web Tools Project

by Boris Minkin

#### FROM THE GROUP PUBLISHER

### As Oracle Recommits to Java, Sun Sweeps 2005 RCAs

by Jeremy Geelan

.....3

#### VIEWPOINT

### Lightweight Java Enterprise Application Frameworks

by Michael Yuan

.....6

#### ENTERPRISE VIEWPOINT

### What CIOs Should Know About Outsourcing Enterprise Java

by Yakov Fain

.....10

#### SOA

### Moving to SOA in J2EE 1.4

*How to take an existing J2EE 1.3 bean and convert it into a Web service endpoint with almost no changes to its business interface*

by Andrei Iltchenko

.....12

#### LIGHTWEIGHT JAVA

### J2EE Lite with Spring Framework

*For a better application*

by Pankaj Tandon

.....40

#### LIGHTWEIGHT JAVA

### Spring and EJB 3.0 in Harmony

*In search of the best of both worlds*

by Aleš Justin

.....48

#### DESKTOP JAVA VIEWPOINT

### Where Are the High-Level Design Open Source Tools?

by Joe Winchester

.....62



**Michael Yuan**  
Guest Editor

## Lightweight Java Enterprise Application Frameworks

**L**ightweight application frameworks are all the rage in the enterprise Java community in the past couple of years. From the pioneering Spring and Hibernate frameworks, to the infusion of technologies like aspect-oriented programming and metadata annotation, to the new standard EJB 3.0 (and Java EE 5.0) specifications, lightweight frameworks have gradually become mainstream. The rise of lightweight technologies was largely due to developers' rebellion against the "heavyweight" of EJB 2.1 (and earlier). Lightweight frameworks aim to make developers more productive and the application less error-prone by removing the rigid EJB 2.1 infrastructure classes/interfaces and excessive XML deployment descriptors (commonly known as "XML hell" in EJB 2.1). Beyond that, lightweight frameworks have also promoted better and cleaner application architectures, and make it easier to reuse business components when you switch vendors.

The core principle shared by all lightweight enterprise Java frameworks is the use of plain old Java objects (POJOs) for the data access and business logic. There are no more infrastructure classes or interfaces to inherit or implement. You just create a POJO to model your data or to implement a business process using the data. Then the POJOs are "wired" together using metadata. For instance, you can wire a POJO to a container service that maps the object to an entry in a relational database; you can wire a POJO method to a transactional service to guarantee the atomicity of its database operations or to a security service to limit access

to the method; you can also wire POJOs together with each other. A key technique in the wiring of POJOs is a design pattern called Dependency Injection (DI). DI uses the lightweight framework container (e.g., a Spring container or an EJB 3.0 container) to inject services or other objects into a POJO. This way, all object instances are created and managed by the container. There is no need for each POJO to manage the life cycle of its service objects or to look up services. DI eliminates a lot of boilerplate code in applications and makes them easier to understand and maintain.



The major differences between lightweight frameworks are how they wire container services together and implement Dependency Injection. The service architecture and metadata expression are the key issues here. In this issue of *JDJ*, Chris Richardson, author of Manning's new *POJO in Action* book, explains

how popular lightweight frameworks such as Spring, Hibernate and EJB 3.0 deliver services to POJOs.

While different lightweight frameworks support different metadata syntax for POJO wiring, developers naturally still want to reuse POJOs across frameworks. In his "Spring and EJB 3.0 in Harmony" article, Aleš Justin, an avid Spring and JBoss user, explores how to write a Spring deployer in JBoss. Using the deployer, you can deploy Spring POJOs side-by-side with EJB 3.0 POJOs in the JBoss Application Server. Very cool!

Now, Spring, Hibernate, and EJB 3.0 are lightweight frameworks for the enterprise middle tier. They are great for implementing business and persistence logic. However, to use

### 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

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

### 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

—continued on page 8

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

[juntao@mail.utexas.edu](mailto:juntao@mail.utexas.edu)



# Innovations by InterSystems



*Rapid development with robust objects*



*Lightning speed with a multidimensional engine*



*Easy database administration*



*Massive scalability on minimal hardware*

## Object Database, Perfected.

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.



Try an innovative database for free: Download a fully functional, non-expiring copy of Caché, or request it on CD, at [www.InterSystems.com/Cache16P](http://www.InterSystems.com/Cache16P)

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

© 2006 InterSystems Corporation. All rights reserved. InterSystems Caché is a registered trademark of InterSystems Corporation. 1-06 CachéInno16JaDeJo

—continued from page 6

them in Web applications, you often still need to provide your own integration code with the servlet/Struts/JSF-driven Web tier. Can we use business POJOs directly in the Web tier? In the Spring world, the Spring MVC framework is a relatively new Web framework that works well with Spring business POJOs. What about the standard-based EJB 3.0 world? Is there a framework that allows developers to write complete Web applications using EJB 3.0–style POJOs and annotations? The answer is the JBoss Seam framework. To understand Seam, let's look at a very simple Hello World application: it has a simple Web page where you can enter your name to “say hello” to Seam. When you click submit, your name is added to the database and the page displays all persons who have said hello so far. The application requires only three simple components. First, write an EJB 3.0 entity bean, the Greeter class, to represent a greeter (i.e., the data model). The `@Entity` annotation tells the container that the Greeter POJO is automatically mapped to a relational database table. The `@Name` annotation specifies that Seam manages the Greeter instances under the name “greeter”. When other application components request an instance of the Greeter bean from Seam, they should use the name “greeter”.

```
@Entity
@Name("greeter")
public class Greeter implements Serializable
{

    private long id;
    private String name;

    @Id(generate=AUTO)
    public long getId() { return id;}
    public void setId(long id) { this.id =
id; }

    public String getName() { return name; }
```

```
public void setName(String name) { this.
name = name; }
}
```

The Web page looks like the following. Notice that the textfield is mapped to the Seam-managed Greeter entity bean via the name “greeter”. When a user enters a value in the text field, Seam automatically creates a Greeter instance encapsulating the user input. When the user submits the form, the event handler method `manager.sayHello()` is invoked by Seam. This method is defined in an EJB 3.0 session bean POJO under the Seam name “manager”, which we will discuss next.

```
<h:form>
Please enter your name:<br/>
<h:inputText value="#{greeter.name}"
size="15"/><br/>
<h:commandButton type="submit" value="Say
Hello"
action="#{manager.sayHello}"/>
</h:form>
```

The event handler session bean, `ManagerAction`, has a Seam name “manager” as described above, so that it can be referenced in the JSF page. Via the `@In` annotation, Seam automatically injects the managed Greeter POJO, which has Seam name “greeter”, into the `ManagerAction` property of the same name. The “greeter” object is already populated with the user input from the JSF page. The `sayHello()` method saves the entity bean POJO to the database and retrieves the current list of greeters, which is out-jected into the Seam context via the `@Out` annotation.

```
@Stateless
@Interceptors({SeamInterceptor.class})
@Name("manager")
public class ManagerAction implements Manager
{

    @In
    private Greeter greeter;
```

```
@Out
private List <Greeter> allGreeters;

@PersistenceContext
private EntityManager em;

public String sayHello () {
    em.persist (greeter);
    allGreeters = em.createQuery("from
Greeter g").getResultList();
    return null;
}
}
```

Since the `allGreeters` list is out-jected into the Seam context, the JSF page can refer to it and display the name list.

```
<p>The following persons have said "hello" to
JBoss Seam:</p>
<h:dataTable value="#{allGreeters}"
var="person">
    <h:column>
        <h:outputText value="#{person.name}"/>
    </h:column>
</h:dataTable>
```

Now, you can see JBoss Seam is a powerful glue that connects EJB 3.0 POJOs and JSF pages. It also expands the dependency injection pattern into a bi-directional injection. As a result, everything is managed by the Seam context and developers can focus on their core business logic, which is the ultimate goal of any lightweight Java framework. In addition, the Seam context also manages HTTP session state and even workflow in external business process engines (e.g., the jBPM engine). Norman Richards, author of *JBoss Developer's Notebook* and *XDoclet in Action*, will tell you more about how Seam makes it super easy to develop stateful Web applications in his article “Seam: The Next Evolution of Web Applications.”

By all indications, lightweight technologies are the future of enterprise Java. Learn it, master it, and use it in your best-of-breed new Internet applications! ☛

“The core principle shared by all lightweight enterprise Java frameworks is the use of plain old Java objects (POJOs) for the data access and business logic”



# 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](http://www.windwardreports.com). You won't believe your eyes.**





**Yakov Fain**  
Enterprise Editor

# What CIOs Should Know About Outsourcing Enterprise Java

**Y**our manager Frank started the meeting by saying that the budget for the new project had been approved, but half of the project will be outsourced to a great team from overseas. Can you imagine, their rates for Java programmers can go as low as \$15 an hour!

No, we're not losing anyone from our team, and you should take it as an opportunity to work as team leaders, helping our new partners to hit the ground running. No, this wasn't my decision; it came from above.

## Three Months Later

**Mary:** I've asked them to add two fields to a JTable on the Invoice screen. The data are being retrieved from our database so they'd need to modify an SQL query as well. I've sent them this e-mail yesterday, but it was night time over there, so they've responded today asking me to send them the modified SQL and write the name of the Java class and method where this new code should reside. I could've done this by myself in two hours.

**Frank:** Just be patient, it's a new application for them. By the way, I'd appreciate it if you could stay a little longer today. We're having a meeting with our colleagues from overseas, but there's a time difference, you know... No worries, they're willing to come to work early, so we're starting our meeting at 7pm.

## Six Months Later

**Frank:** The system has to go to UAT in two weeks. We've all worked hard, our remote colleagues put in lots of overtime. John, you're our Java expert, and you've spent the last two weeks doing the code review of that module. Why does it work a little slow?

**John:** Well, that module isn't written in Java. I mean, they were using Java syntax,

but it wasn't Java programming. There are chunks of unused code fragments, the code isn't object-oriented, they used the wrong Java collections, and there's unnecessary synchronization all over the place. But I can re-write the entire piece in three weeks.

**Frank:** OK, let's do it – but quietly.

After spending many nights in the office, the project was saved. Frank got promoted for delivering the project almost on time and showing “strong leadership in managing cost-saving external resources.” But the team's morale went down the drain; two local resources (a k a John and Mary) got small bonuses and started looking for new jobs.

## Post-Mortem Analysis

Unfortunately, more and more CIOs believe that computer programming is a commodity skill that can be bought cheaply when needed and replaced easily like a receptionist, mailman, or any other clerk. They don't believe that having a pool of knowledgeable and talented developers adds any value to the organization. This wouldn't be the case if the development managers (the Franks) explained to them the price that's paid for the success of such projects. But most of these managers never do this, because of conflict of interest:

Frank's only goal is his smooth movement up the corporate ladder. Moreover, to increase his importance, Frank inflates the resources needed for the project on purpose. The CIO doesn't have the budget for several additional \$70K-a-year developers, so he settles on the same number of \$30K developers from overseas with similar résumés. Realistically, the “cheap” labor is actually an additional expense on top of the salaries of local employees.

Another hidden expense is the extra time spent writing super-detailed functional specifications and validating the overseas

work. Here's one more: for security reasons, you may have to create and maintain a separate encrypted version of your database for the offshore team.

Having said all this, I can't blame the overseas developers. Their countries are experiencing a golden IT rush, so young kids are ready to dive into muddy Java waters after spending several months in vocational school (if I were in their shoes I'd do the same thing). They put in long hours trying to learn programming and the business of their rich clients (not to be confused with “fat clients”) on the run. As a result of this IT boom, the turnover rate in offshore teams can be as high as 100%. You can often see it just by looking at the source code. Sometimes you get a feeling that a 200-line Java program was written by 10 different people of different qualifications. Forget about naming conventions, design patterns, or any programming style.

Hey, Frank, if you need seven people for a project, have the guts to say seven and not 10. Yes, you won't have a chance to manage an international (or as they like to say global) project, but you'll definitely sleep better at night. Before giving a chunk of your project to a company overseas, talk to your developers and ask them if they really need this help. Your developers are human beings and not just nameless resources.

On the other hand, outsourcing works fine for small businesses because both parties know that the owners of such businesses count their money and won't pay for poor-quality jobs. It also works when you hire an offshore team of senior people who know the business you're in. No, their rates aren't cheap, and don't have to be! But such teams usually consist of professionals, who take pride in their work, deliver on time without putting an extra burden on your own developers, and even mentor your staff. This is the outsourcing I vote for, but I'm not the CIO of your company. ♦

“Unfortunately, more and more CIOs believe that computer programming is a commodity skill that can be bought cheaply”

**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](http://smartdataprocessing.com)) and several chapters for *Java 2 Enterprise Edition 1.4 Bible*. He leads the Princeton Java Users Group.

[yakovfain@sys-con.com](mailto:yakovfain@sys-con.com)



# Best AJAX Toolkit\*

## TIBCO General Interface

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



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

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

 **TIBCO®**  
The Power of Now®

# Moving to SOA in J2EE 1.4

by Andrei Iltchenko

*How to take an existing J2EE 1.3 bean and convert it into a Web service endpoint with almost no changes to its business interface*

**T**his past year the J2EE 1.4 specification enjoyed increased adoption by the industry, with most major application server vendors having released their J2EE 1.4 products and more and more enterprises upgrading their application servers and production applications to comply with it.

Apart from merely upgrading, there's a natural desire to leverage the primary focus of J2EE 1.4 — its support for service oriented architectures and Web services — and make it benefit the business by exposing the business logic of J2EE applications in a way that makes them portably accessible from the various heterogeneous environments that are around today.

Here at Compuware (the company I work for) we chose to make the ability to generate J2EE 1.4 applications one of the top requirements of the upcoming version of our OptimalJ product, an MDA-based development product that enables you to generate (from user-defined models) complex, standards-compliant J2EE 1.3 and J2EE 1.4 applications that run on JBoss, WebSphere, and WebLogic. To do this, we were faced with the formidable task of ensuring that the user-modeled enterprise beans or Stateless Session Beans (SLSBs) that used to be translated into EJB 2.0 code can now be translated into EJB 2.1 Beans with Web Service endpoints with a minimum of changes to the signatures of their business methods. The same was accomplished for Java components running in the Web container.

Essentially what we learned is how to take an existing J2EE 1.3 bean (a SLSB or a Java component running in the Web container) and convert it into a Web Service endpoint with almost no changes to its business interface by craftily applying Java-to-WSDL/XML mapping to its business interface.

What makes this difficult is that when a bean is exposed and ac-



cessed as a Web Service endpoint, parameter passing is subject to SOAP serialization rules, which are far more restrictive than RMI-IIOP serialization semantics.

What we did was stress the ASs' compliance to J2EE 1.4 to its limits and gained firsthand knowledge of what works and what doesn't. We found numerous problems in the application services that we targeted — JBoss 4.0.x and WebSphere 6.0.x, which we addressed in cooperation with the respective vendors. In this article I will share the valuable knowledge that we learned on this project, show portable solutions, and draw attention to common pitfalls.

## Service Endpoints in J2EE 1.4

Web Service endpoint is a term used in J2EE to describe Web Service components deployed in a J2EE container. A service endpoint can be implemented using a stateless session bean, in which case it runs in the EJB container, or as a Java class that is registered as a servlet, in which case it runs in the Web container and is termed a JAX-RPC endpoint.

Associated with each service endpoint is its service endpoint interface (SEI for brevity). Given that the Web Service endpoint is implemented by a J2EE component, it's sometimes called a service implementation bean.

Conceptually and visually the SEI is very similar to the remote component interface of Enterprise JavaBeans. It too extends `java.rmi.Remote`, its methods must declare `java.rmi.RemoteException` in their throws clauses, and looking at the code of a SEI and a remote component interface alone is often impossible to tell the two apart. This similarity is delusive since, in the case of the SEI, parameter and return value passing is subject to the restrictive SOAP serialization rules, while for the remote component interface the more liberal Java serialization semantics apply (let alone communication via local component interfaces where no extra restrictions apply).

SOAP serialization rules only permit argument and return values of the following types:

1. The primitive Java types (with the exception of `char`) and their wrapper counterparts
2. The standard JDK classes `java.lang.String`, `java.util.Date`, `java.util.Calendar`, `java.math.BigInteger`, `java.math.BigDecimal`, `javax.xml.namespace.QName`, and `java.net.URI`
3. JavaBeans with default constructors whose properties and public fields types are, in turn, allowed SOAP serialization types
4. Java arrays whose underlying types are, in turn, permitted SOAP serialization types. An important special case is that arrays of type `Object` aren't supported because `java.lang.Object` is not enumerated as a supported type. Even though not explicitly mentioned in the specification, multidimensional arrays also pose problems and should be avoided to achieve compliancy with the WS-I Basic Profile.

And that's all — no Java collections, even if a `JavaBean` class implements



Andrei Iltchenko is a development lead at Compuware Corporation where he works on the MDA product OptimalJ and is responsible for the business logic area of OptimalJ-generated J2EE applications. He is also a Sun certified Java developer for Java Web Services, a Sun Certified Business Component Developer, a Sun Certified Developer, and a Sun Certified Programmer.

andrei.iltchenko@nl.compuware.com



java.io.Serializable, it doesn't imply its eligibility for SOAP serialization! Even though there's a concept of custom serializers/deserializers for Web Service endpoints, such serializers aren't portable across Web Services for J2EE providers and, as a result, aren't supported in J2EE 1.4. Yet, despite the apparent restrictions, we found out that in many situations it's still possible to devise portable ways of exposing your business interfaces as SEIs without changing them much.

An SEI can coexist with the remote/local component interfaces of a SLSB or the business interface of a Web component in which case it may have a different number of business methods than the accompanying component interface(s). Like its remote/local counterparts, a SEI can feature overloaded methods that should be mapped to wsdl:operations with distinct names (more on that can be found in the Pullout on WS-I Basic Profile). We observed that JBoss didn't support overloaded methods in SEIs prior to version 4.0.4. (See <http://jira.jboss.com/jira/browse/JBWS-463>.)

Another important difference between the remote or local component interface and the SEI is that the latter must be accompanied by a WSDL definition that provides a canonical description of its Web Service endpoint that can be published to third parties (which may not necessarily be using a Java environment).

This essentially brings in an additional dimension because the SEI's WSDL definition influences how method invocations are serialized into XML. One way to influence the serialization of method invocations is the binding that the WSDL definition has assigned to the SEI. J2EE 1.4 supports two portable bindings — document-literal and rpc-literal, whose details can be found in [1, 2, and 3]. Thus in J2EE 1.4 a Web Service endpoint is fully described by the pair { SEI, WSDL } and the same SEI accompanied by different WSDL definitions leads to different XML messages on the wire.

The fact that a J2EE Web service endpoint is defined by the { SEI, WSDL } pair entails two different ways of developing one:

- *Start from a SEI.* The rules of this approach are defined in the Java-to-XML/WSDL mapping whose definition can be found in the JAX-RPC 1.1 specification.
- *Start from a WSDL definition.* This development approach is governed by the WSDL/XML-to-Java mapping.

The two mappings aren't symmetrical. Moreover each allows variations. That is, the transformation  $SEI_1 \rightarrow WSDL \rightarrow SEI_2$  need not imply that  $SEI_1 = SEI_2$ . When dealing with an existing stateless session bean or existing business logic component running in the Web container, it is of course the Java-to-XML/WSDL mapping that's applicable. This mapping is also prevalent in new projects and the rest of the article will concentrate on it. (The WSDL/XML-to-Java mapping is typically turned to only when one needs to develop a Web Service endpoint that conforms to the interface and binding information provided by a given WSDL document.)

## Java-to-XML/WSDL Mapping

When using the development approach based on the Java-to-XML/WSDL mapping, what you do first is define the SEI for the component you want to expose as a Web Service. You typically do so by copying the signatures of an existing component's business methods from its remote component

interface (for EJBs) or business interface (for components running in the Web container). Sometimes you may have to change the signature slightly and I will explain later when that is necessary.

When mapping Java types to XML Schema definitions all non-standard types are regarded as JavaBeans. An essential and little considered aspect of this mapping is that only public non-final non-transient fields and read/write properties as defined by the `java.beans.Introspector` class are taken into account. Java exceptions are treated slightly differently; in their case read properties are also counted in. Each read property must be accompanied by a corresponding parameter in the constructor; otherwise it wouldn't be possible to assign the property a value. We found out that some application servers (e.g., JBoss 4.0.x — cf. <http://jira.jboss.com/jira/browse/JBAS-2139>) have problems with read-only properties in exceptions when a property's type is an array.

The fact that 1) public non-final fields are very uncommon and their use is discouraged and that 2) properties are queried for by using `java.beans.Introspector` means that we can accompany “problematic” Java types — that in themselves don't qualify for SOAP serialization rules — with `BeanInfo` counterparts that present explicit information of what the properties of a given Java type are. Listing 1 offers an example of a “problematic” Java type that was used as a superclass for all application exceptions in our project.

The problem with this class is that it features two properties that don't qualify for SOAP serialization — cause of type `Throwable` and values of type `Object[]`. No matter how you map this class to an XML Schema complex type (even if you exclude the problematic properties from the complex type's content model), you're bound to run into problems during your application's execution. Listing 2 shows an accompanying `BeanInfo` class that makes `AlturaException` a legitimate type for SOAP serialization.

As you can see the `BeanInfo` class simply excluded the cause property from the serialized state. It also redirected the values property to a new getter and setter — `getStringValues` and `setStringValues` (the setter is required to cater to those ASs that don't yet adequately support read-only properties of array types in exceptions). The new getter and setter are shown below:

```
/** Returns a String representation of this object's arguments. */
public String[] getStringValues() {
    return getValues() == null
        || getValues().getClass() != String[].class
        ? null : (String[]) getValues();
}

public void setStringValues(String values[]) {
    args = values;
}
```

When dealing with abstract Java classes, some application servers (e.g., WebSphere 6.0.x) require that they be mapped to abstract complex types, which is a sensible requirement that should be followed not only for portability but also to support polymorphism in the SEI methods.

With our changes to the serialized state of `AlturaException`, we can now portably map it to an XML Schema complex type:

```
<xsd:complexType name="AlturaException" abstract="true">
```

```
<xsd:sequence>
  <xsd:element name="key" type="xsd:string"/>
  <xsd:element name="values" type="xsd:string" nillable="true"
    minOccurs="0" maxOccurs="unbounded"/>
</xsd:sequence>
</xsd:complexType>
```

In applying the Java-to-XML/WSDL mapping it's worth keeping in mind that certain application servers like WebSphere 6.0.x have a code generation and compilation step at service endpoint deployment time. WebSphere, for example, uses this step to generate serializers, deserializers, and helper classes for all JavaBean classes used (directly or indirectly) in a SEI. An essential aspect is that during this step WebSphere applies the WSDL/XML-to-Java mapping to the XML Schema complex type definitions of each endpoint being deployed and verifies whether the resulting property's types are the same. That's a given:

1.  $XML_T(T)$ , the XML-to-Java mapping function whose domain is JavaBean classes that qualify for SOAP serialization rules; and
2.  $Java_C(C)$ , the Java-to-XML mapping function whose domain is XML Schema complex types; and
3. A JavaBean type called Bean that has a property called prop; and
4. A JavaBean type  $Bean' = Java_C(XML_T(Bean))$

WebSphere requires that the Bean.prop type be equal to that of Bean.prop. (As mentioned above, the composite function  $Java_C(XML_T(T))$  need not be the identity function for T.) Given that this rule is applied by WebSphere only at the JavaBean level, we discovered that it's quite easy to satisfy it. What you have to watch out for are properties whose type is a primitive Java type, a wrapper class, or an array thereof. You'll then have to be careful about using the nillable attribute when mapping them to XML Schema local elements. For example, given this JavaBean:

```
public class ExampleBean {
    public Double[] getDoubleValues() {}
    public void setDoubleValues(Double[] p) {}
    public double getFirstValue() {}
    public void setFirstValue(double p) {}
}
```

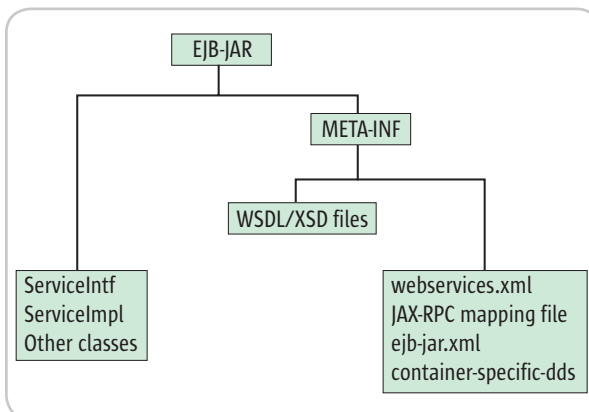


Figure 1 New DDs packaged in an ejb-jar module

Use the following XML Schema mapping:

```
<xsd:complexType name="ExampleBean">
  <xsd:sequence>
    <xsd:element name="doubleValues" type="xsd:double"
      nillable="true"
      minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element name="firstValue" type="xsd:double"
      nillable="false"/>
  </xsd:sequence>
</xsd:complexType>
```

Even if you're not targeting WebSphere, I still recommend that you follow this practice to increase your application's portability. (Some additional details on the application of the Java-to-XML/WSDL mapping are presented in the pullout on the WS-I Basic Profile.)

## Standard J2EE 1.4 Deployment Descriptors for Service Endpoints

Besides SEIs and WSDL definitions, J2EE defines two new deployment descriptors that must be present in every ejb-jar file and every WAR file that contain Web Service endpoints: the Web Services Deployment Descriptor (one per J2EE module with endpoints must be named Webservices.xml) and the JAX-RPC Mapping Deployment Descriptor (one or more per J2EE module can have an arbitrary name). (For full details on these deployment descriptors, see Web Services for J2EE 1.1 specification.) Here I will discuss some crucial aspects of the two DDs that aren't given much attention in the specification.

The specification committee did a very good job in ensuring that the syntax of the deployment descriptors is the same regardless of whether they reside in a WAR of an ejb-jar file. The committee also took into account the bitter experience of past J2EE specifications to ensure that the metadata provided by these DDs is extensive enough not to necessitate application server-specific extensions. For example, in the case of JBoss AS 4.0.x, the two DDs are all you have to provide in most situations to ensure a successful endpoint deployment.

Figure 1 shows how the new DDs are packaged in an ejb-jar module.

The syntax of the webservices.xml deployment descriptor is rather simple (for an example of this DD, refer to the sample application accompanying this article that can be downloaded from the resource center).

At this stage it's worth noting that J2EE 1.4 allows a single Web Service endpoint to be associated with multiple target URLs that it will serve. This is useful when the endpoint can be accessed via different transports (e.g., HTTP and HTTPS). Each such concrete URL location is termed a J2EE port and maps directly on the wsdl:port concept, implying that there's a one-to-one correspondence between J2EE ports and wsdl:ports in a given module.

Each J2EE port is assigned a name in the Webservices.xml DD, which need not be the same as its corresponding wsdl:port name and must be unique among all ports declared in the containing J2EE module. Thus a J2EE port is a combination of a Web Service endpoint and a wsdl:port URL location and is defined by the { endpoint, wsdl:port URL } pair.





60 minutes!

That's all it takes to deploy **IntelliVIEW**

IntelliVIEW makes enterprise reporting-server deployment easier than ever before. You can install, configure the server and even have your first report out, all within 60 minutes! The 'IntelliVIEW Intelligent Installer' automatically checks for all prerequisites and also takes care of all essential web/ app server and database configurations.

**The IntelliVIEW advantage:**

- Connects to multiple databases
- Extremely quick report creation
- Range of report outputs PDF, Excel, HTML...
- Scheduling & emailing facility
- 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](mailto:sales@intelliview.com)

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

 product of  
**Synaptris**

It's when you apply the Java-to-XML/WSDL mapping to your SEIs that you have to decide on how many URL locations you want each endpoint to serve (typically one) and create wsdl:port definitions for each of them.

What the Webservice.xml DD does is declare all ports present in its J2EE module. For each port the DD specifies:

1. The port's name
2. The name of the corresponding wsdl:port
3. The port's Web Service endpoint (service implementation bean)
4. The WSDL definition accompanying the endpoint
5. The JAX-RPC mapping file that contains Java <-> XML mapping metadata necessary to tell the container how to serialize/deserialize method invocations on the SEI

The JAX-RPC mapping DD, on the other hand, is far more involved and is rather complex. It spells out:

1. How the methods of the SEI are mapped to their corresponding wsdl:operations.
2. The correlation between the JavaBeans and XML Schema complex types that are used (possibly indirectly) by the SEI's methods (a java-xml-type-mapping must be provided for every such JavaBean). The primitive Java types, their wrapper counterparts, and arrays in generally don't require a mapping. This metadata is needed since the container may not have knowledge of what JavaBean was mapped to what XML Schema complex type during the application of the Java-to-XML mapping.
3. For JavaBeans that represent exceptions, it provides information on how the element content of the exception's complex type is related to the arguments of the exception's constructor (an exception mapping must be provided for every Java exception in addition to a java-xml-type mapping that correlates the exception with the corresponding complex type).

The trickiest parts of this DD are: 1) the part that maps a SEI method's parameters and return type to wsdl:parts and 2) one that correlates JavaBeans with XML Schema complex types. This information is used by the container in creating/configuring serializers/deserializers for each { JavaBean, XML complex type} pair during application deployment.

The JAX-RPC Mapping DD lets us specify mapping for anonymous XML Schema complex types. (Anonymous complex types may be obtained when applying the Java-to-XML/WSDL mapping to your SEI, although it's always possible to avoid them by using exclusively root-level types.) We learned that java-xml-type mappings supplied for anonymous complex types don't work on some application servers like JBoss 4.0.x. You would thus be better off steering clear of using anonymous types when applying the Java-to-XML mapping to ensure your application's portability.

If your SEI makes use of JavaBean classes that form an inheritance hierarchy, then a java-xml-type mapping must be specified for every class that participates in it (excluding java.lang.Object and, when dealing the JavaBeans that are used as exceptions, java.lang.Exception). Each java-xml-type mapping only has to list the properties its JavaBean introduces and omit any inherited ones.

The mapping of a SEI method's parameters and return type has a peculiar trait (that's not explicitly mentioned in the specification) in that some additional steps, which are explained below, have to be taken to support parameters and return values whose types are either interfaces or abstract classes (i.e., classes that can't be instantiated). This is a big difference from the remote/local component interface whose methods are free to use non-instantiatable types in their signatures without requiring any additional work by the developer. If the extra work required to support non-instantiatable types in an existing business method isn't an option, you'll have to change the method's signature (if the return type isn't instantiatable, you may have to change the method's name).

The same problem occurs when dealing with JavaBeans with properties whose types can't be instantiated; they too require additional work for the same reason.

If you still have to have parameters/return types/JavaBean properties of non-instantiatable types, you'll have to take the following additional actions:

1. When applying the Java-to-XML mapping and dealing with an abstract JavaBean referenced in a SEI method (possibly indirectly), it has to be mapped to an abstract XML Schema complex type. Even if the SEI doesn't reference a concrete JavaBean sub-classing the abstract one, you must map at least one such concrete type to XML Schema. For instance, given the following SEI:

```
public interface DemoServiceEndpoint extends java.rmi.Remote {
    public AbstractDescr processDescr(AbstractDescr p) java.rmi.
    RemoteException;
}
```

where the business method is implemented as:

```
public AbstractDescr processDescr(AbstractDescr p) {
    System.out.println("The description is: " +
    p.getDescription());
    AbstractDescr returnValue = new Descr();
    returnValue.setDescription("Works as a return type!");
    return returnValue;
}
```

and the following JavaBeans classes:

## Lightweight Business Logic Components

In many J2EE Web applications (including those that can be generated with Compuware's Optimal product) the middle tier is implemented using non-EJB business logic components that are plain Java classes that implement a particular business interface (analogous to the component interface of EJBs), the J2EE DAO pattern being a typical example of such a class.

Such components don't require the managed environment needed for EJBs and can run in the Web container or in the application client container. By running in a J2EE container they have access to programmatic global transaction management using JTA, local transaction management with JDBC, connection pooling, as well as JMS-related services.

In J2EE 1.3 and before, architectures whose middle tier was built on such components were inherently non-distributed because the business logic components were always running in the same JVM that presentation or client code was executing — a severe scalability limitation that necessitated turning to EJBs for some J2EE projects. With the advent of J2EE 1.4 and Web Service endpoints, such components are becoming distributed, further aiding lightweight non-EJB architectures.



```

abstract public class AbstractDescr {
    private String description;
    public String getDescription() {
        return description;
    }
    public void setDescription(String value) {
        description = value;
    }
    ...
}
// Concrete class not referenced in the SEI
public class Descr extends AbstractDescr {
    ...
}

```

A correct way of mapping the SEI to XML would be something along these lines:

```

<xsd:complexType name="AbstractDescr" abstract="true">
    <xsd:all>
        <xsd:element name="description" type="xsd:string"/>
    </xsd:all>
</xsd:complexType>
<!-- Mapped from a Java type not referenced in the SEI -->
<xsd:complexType name="Descr">
    <xsd:complexContent>
        <xsd:extension base="tns:AbstractDescr"/>
    </xsd:complexContent>
</xsd:complexType>

```

2. Then you'll need to provide a java-xml-type mapping for both AbstractDescr and Descr in the corresponding JAX-RPC mapping file:

```

<java-xml-type-mapping>
    <java-type>packageName.AbstractDescr</java-type>
    <root-type-qname xmlns:typeNS="http://packageName">typeNS:
AbstractDescr</root-type-qname>
    <qname-scope>complexType</qname-scope>
    <variable-mapping>
        <java-variable-name>description</java-variable-name>
        <xml-element-name>description</xml-element-name>
    </variable-mapping>
</java-xml-type-mapping>
<java-xml-type-mapping>
    <java-type>packageName.Descr</java-type>
    <root-type-qname xmlns:typeNS="http://packageName">typeNS:Descr</
root-type-qname>
    <qname-scope>complexType</qname-scope>
</java-xml-type-mapping>

```

The reason why it's necessary to map a non-instantiatable JavaBean as an abstract XML Schema complex type is to force the use of the xsi:type attribute to indicate a derived type during the serialization of a method invocation/return. The xsi:type attribute will then appear on soap:Body elements corresponding to abstract parameters or properties. The rationale for also mapping at least one concrete JavaBean sub-classing each abstract one is to let non-J2EE 1.4 clients correctly invoke a SEI method making use of abstract types.

In the case of the example above, the following request and response envelope will be generated by the endpoint client (running on .NET) and the application server (WebSphere in this example):

#### Client request envelope

```

<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope
    xmlns:mrns0="http://packageName"
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:tns="http://packageName"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soap:Body>
        <mrns0:processDescr>
            <p xsi:type="tns:Descr">
                <tns:description>Sample description</tns:description>
            </p>
        </mrns0:processDescr>
    </soap:Body>
</soap:Envelope>

```

#### Server response envelope

```

<soapenv:Envelope
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soapenv:Header/>
    <soapenv:Body>
        <p558:processDescrResponse xmlns:p558="http://SEIpackageName">
            <processDescrReturn xsi:type="p287:Descr"

```

### WS-I Basic Profile

Not surprisingly, given the popularity of Web Services, various open standards and specifications that relate to the technology abound. Some specifications are under-specified or leave a lot of room for interpretation; others contain inconsistencies or errors. This resulted in portability issues in the industry a few years ago. To combat that the WS-I consortium was established that is chartered to promote Web Services interoperability across platforms, applications, and programming languages. Probably the most important contribution of WS-I to date is its WS-I Basic Profile specification. J2EE 1.4 requires that compliant application servers conform to the Basic Profile Version 1.0.

The Basic Profile contains numerous clarifications, corrections, additions, and examples pertaining to the core Web Services specifications — WSDL 1.1, SOAP 1.1, and UDDI 2.0. Most importantly it demarcates the subset of features whose use is guaranteed to be portable. At the moment, the Basic Profile is at version 1.1, which is based on version 1.0 and contains errata against that specification.

Despite the required compliance to the Basic Profile on your application server's part, it's still necessary to follow its rules when developing your J2EE Web Service endpoints since not everything can be taken care of by the application server. The most important examples of constructs prohibited by the profile that should be kept in mind when developing J2EE Web Service endpoints by using the Java to XML/WSDL mapping are:

1. Refrain from using the encoded bindings and SOAP encoded arrays, since are explicitly banned by the profile;
2. Not using overloaded wsdl:operations in a wsdl:portType or wsdl:binding. As a result, when applying the Java-to-WSDL mapping to a SEI, the SEI's overloaded methods (if any) must be mapped to wsdl:operations with distinct names;
3. If you choose the document-literal binding for a given SEI, you should make use of the wrapper style (more details on the wrapper style can be found in Section 6 of JAX-RPC 1.1) whereby all parameters of a given SEI operation are wrapped inside a complex type rather than being listed as wsdl:parts. This is because the profile allows at most one wsdl:part in a wsdl:message referred to from a document-literal binding.

```

xmlns:p287="http://packageName">
  <p287:description>Works as a return type!</p287:description>
</processDescrReturn>
</p558:processDescrResponse>
</soapenv:Body>
</soapenv:Envelope>

```

We've tested the use of abstract types in SEI methods pretty extensively on WebSphere 6.0.x. In JBoss, support for such types was added in the recent version of the application server — 4.0.4 (cf., <http://jira.jboss.com/jira/browse/JBWS-349>) and doesn't yet appear to work for JavaBean properties (cf., <http://jira.jboss.com/jira/browse/JBAS-2591>). You'd be better off avoiding abstract types in SEI methods for the sake of your application's portability and its ease of use.

### Solutions to Using Objects of Common Java Classes

This section provides examples of intricate constructs and presents portable ways of dealing with them.

#### Java Collections

Java collections are ubiquitous in existing enterprise applications and not being able to use them in SEIs might pose a serious limitation. This section presents a portable solution to being able to use collections in your Web Service endpoints. I'll use the following method declaration as an example throughout this section: `public java.util.Collection retrieveAllCustomers()` throws `RemoteException`;

The declaration was taken from the remote interface of an existing session bean. This business method returns a collection of `CustomerTransferObjects` (a.k.a. Value objects) representing all customer records in a persistent store. As we learned in the previous section, the first thing about this declaration that we should correct to avoid potential problems is that it features an interface as a return type (because the return type doesn't make up part of the signature of a Java method, we'll need to change the method's name too): `public java.util.ArrayList retrieveAllCustomersSei()` throws `RemoteException`.

This looks better, yet all standard Java collection types in the JDK don't qualify for SOAP serialization per se. However, nothing prevents us from deriving our own collection class from `ArrayList` (or any other `Collection` implementation for that matter) and then presenting it as a simple JavaBean with a single array type property with the help of an accompanying `BeanInfo` class. Listings 3 and 4 show what we can do:

With the help of the accompanying `BeanInfo` class and the `getTypedElements/setTypedElements` getter/setter, `TransferObjectList` has become a simple JavaBean (that qualifies for SOAP serialization rules) with one array-type property `typedElements`. We can now proceed to apply the Java-to-XML mapping to the JavaBean to get the corresponding XML Schema definition:

```

<xsd:complexType name="TransferObjectCollection">
  <xsd:sequence>
    <xsd:element name="typedElements"
      type="tns:CustomerTransferObject" nillable="true"
      minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>

```

Now we have to tell the container that the `TransferObjectList` Java class is mapped to the `TransferObjectCollection` XML Schema complex type by providing a corresponding java-xml-type mapping in the module's JAX-RPC mapping DD:

```

<java-xml-type-mapping>
  <java-type>example.TransferObjectList</java-type>
  <root-type-qname xmlns:typeNS="http://example">typeNS:TransferObjectCollection</root-type-qname>
  <qname-scope>complexType</qname-scope>
  <variable-mapping>
    <java-variable-name>typedElements</java-variable-name>
    <xml-element-name>typedElements</xml-element-name>
  </variable-mapping>
</java-xml-type-mapping>

```

#### Data Transfer Objects

The Data Transfer Object (DTO for short) is a popular J2EE pattern that lets one minimize the amount of remote calls when communicating with the middle tier of an enterprise application. A DTO is a serializable Java class containing the data necessary in the context of the current use — e.g., updates to the CMP fields of an entity bean. What makes DTOs different from the cases we've dealt with so far in using `BeanInfo` classes is that DTOs frequently feature inheritance, dealing with which warrants a separate explanation. Listing 5 shows two Transfer objects forming up an inheritance hierarchy.

The problem with each of these two classes is that because of the `blobRetriever` property, none of them qualifies for SOAP serialization rules. Because `PersonTransferObject` is abstract, it might seem sufficient to accompany only `CustomerTransferObject` with a `BeanInfo` class. Especially since `java.beans.Introspector`, if it finds a `BeanInfo` class for a JavaBean, regards the explicit `BeanInfo` information as being representative of the bean's superclasses as well. So one might want to provide a `BeanInfo` class for `CustomerTransferObject` that lists its properties along with the properties of the superclass (note that the `BeanInfo` class also changes the property name from `lastName` to `name`). See Listing 6.

Given that:

1. Both Java types are mapped to XML Schema complex types:

```

<xsd:complexType name="PersonTransferObject" abstract="true">
  <xsd:sequence>
    <xsd:element name="key" type="xsd:string"/>
  </xsd:sequence>
</xsd:complexType>

```

### Why RMI-IIOP/RMI-JRMP and SOAP Serialization Semantics Are Different

A cardinal characteristic of RMI-based protocols is that they use standard Java object serialization, which serializes/deserializes objects into/from a binary stream. This enables it to support objects of all classes that implement the `java.io.Serializable` interface.

SOAP serialization rules, employed in communication with Web Service components, on the other hand, marshal/unmarshal objects into an XML representation that conforms to a particular XML Schema definition. Coupled with the fact that the main idea behind Web Services is platform-independence and interoperability, this trait means that for each Java type there must be agreement in the industry on how to translate its objects into XML. If we consider a `java.util.Collection` object, for instance, behind the scenes we may be dealing with a linked list, a binary tree, a hash table, or a simple array. In each of these cases the object's XML representation may have to differ.



Complex JAVA J2EE Hosting made easy.

**WebAppCabaret<sup>sm</sup>**

<http://www.webappcabaret.com/jdj.jsp>  
**1.866.256.7973**

## JAVA J2EE-Ready Managed Dedicated Hosting Plans:

### Xeon III

**SAMEDAY  
SETUP**

Dual 3.2 GHz Xeons  
4GB RAM  
Dual 73GB SCSI  
1U Server  
Firewall  
Linux  
Monitoring  
NGASI Manager

**\$399**  
monthly

### Pentium 4 I

**SAMEDAY  
SETUP**

**FREE  
SETUP**

2.4 GHz P4  
2GB RAM  
Dual 80GB ATA  
1U Server  
Firewall  
Linux  
Monitoring  
NGASI Manager

**\$199**  
monthly  
2nd month  
FREE

### 4Balance I

1 Database Server  
and 2 Application  
Servers connected  
to 1  
dedicated load  
balancing device.  
Dual Xeons.  
High-Availability.

**\$1724**  
monthly

**NEW!** Geronimo 1.0 Hosting . Virtual Private Servers(VPS) starting at \$69/month

At **WebAppCabaret** we specialize in **JAVA J2EE Hosting**, featuring **managed dedicated servers** preloaded with most open source JAVA technologies.  
**PRELOADED WITH:**

JDK1.4 . JDK1.5 . Tomcat . Geronimo . JBoss . Struts . ANT . Spring . Hibernate  
Apache . MySQL . PostgreSQL . Portals . CRM . CMS . Blogs . Frameworks  
All easily manage via a web based control panel.

#### Details:

- All Servers installed with the latest Enterprise Linux
- Firewall Protection
- Up to 60 GB daily on site backup included at no extra charge per server.
- Database on site backup every 2 hours
- Daily off site database backup
- A spare server is always available in case one of the server goes down
- Intrusion detection.
- 24x7 Server and application monitoring with automatic self healing
- The Latest Bug fixes and Security updates.
- Tier 1 Data Center. 100% Network Uptime Guarantee
- Guaranteed Reliability backed by industry-leading Service Level Agreements

Log on now at <http://www.webappcabaret.com/jdj.jsp> or call today at  
**1.866.256.7973**



**WebAppCabaret<sup>sm</sup>**

## JAVA J2EE Hosting

Prices, plans, and terms subject to change without notice. Please log on to our website for the latest price and terms. Copyright © 1999-2006 WebAppShowcase • All rights reserved • Various trademarks held by their respective owners.



```

        <xsd:element name="name" type="xsd:string"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="CustomerTransferObject" abstract="true">
    <xsd:complexContent>
        <xsd:extension base="tns:PersonTransferObject">
            <xsd:sequence>
                <xsd:element name="title" type="xsd:string"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

```

2. A java-xml-type mapping is provided to correlate the JavaBeans with their complex type mappings:

```

<java-xml-type-mapping>
    <java-type>example.PersonTransferObject</java-type>
    <root-type-qname xmlns:typeNS="http://example">typeNS:
PersonTransferObject</root-type-qname>
    <qname-scope>complexType</qname-scope>
    <variable-mapping>
        <java-variable-name>key</java-variable-name>
        <xml-element-name>key</xml-element-name>
    </variable-mapping>
    <variable-mapping>
        <java-variable-name>name</java-variable-name>
        <xml-element-name>name</xml-element-name>
    </variable-mapping>
</java-xml-type-mapping>
<java-xml-type-mapping>
    <java-type>example.CustomerTransferObject</java-type>
    <root-type-qname xmlns:typeNS="http://example">typeNS:
CustomerTransferObject</root-type-qname>
    <qname-scope>complexType</qname-scope>
    <variable-mapping>
        <java-variable-name>title</java-variable-name>
        <xml-element-name>title</xml-element-name>
    </variable-mapping>
</java-xml-type-mapping>

```

#### Listing 1: AlturaException.java

```

public abstract class AlturaException extends Exception {
    /** A key in a resource bundle file. */
    private String key;
    /** Parameters for the error message. */
    private Object[] args = new Object[0];
    private Throwable cause = this;

    public AlturaException(String key) {
        this.key = key;
    }
    public AlturaException(String key, Object[] args) {
        this(key);
        this.args = args;
    }
    public AlturaException(String key, Object[] args, Throwable
cause){
        this(key, cause);
        this.args = args;
    }
    public String getKey() {

```

This setup will work on some application servers but will fail on others. To ensure portability, a BeanInfo class should be provided for the abstract superclass too, especially if the subclass's BeanInfo class changes the accessor methods associated with the properties defined in a base class. Listing 7 shows a BeanInfo class for the abstract superclass that produces a portable deployment.

## Conclusion

The J2EE 1.4 specification includes enough provisions to ensure that the business logic of enterprise applications can be exposed to the world by way of Web Service components in a secure fashion.

Throughout our project we put support for J2EE 1.4 features to the test in two major application servers and discovered that the vendors did a good job in implementing the specification for the average usage, but when taking the spec to the max for enterprise production environments, quite a number of omissions/bugs/limitations were discovered and fixed in cooperation with respective vendors.

All the solutions presented here have been verified to work on JBoss 4.0.4 and WebSphere 6.0.2.5. They are also employed in our upcoming OptimalJ product, where they are transparently put to use when generating J2EE applications from user-supplied UML models.

## Acknowledgments

I would like to express my gratitude to Paulo Moreira with whom I worked on creating support for this functionality in our OptimalJ product. I am also thankful to Phil Adams of IBM WebSphere development and his team for the valuable feedback they provided on the article and to all the individuals at JBoss, Inc. who carefully reviewed it.

## References

1. Web Services Description Language (WSDL) 1.1: <http://www.w3.org/TR/wsdl>
2. Java API for XML-based RPC (JAX-RPC) 1.1: <http://java.sun.com/xml/downloads/jaxrpc.html#jaxrpcspec>
3. WS-I Basic Profile 1.1: <http://www.ws-i.org/Profiles/BasicProfile-1.1.html>
4. WebServices for J2EE 1.1 (JSR-921): <http://www.jcp.org/en/jsr/detail?id=921>

```

        return key;
    }
    public Object[] getValues() {
        return args;
    }
    public Throwable getCause() {
        return (cause==this ? null : cause);
    }
}

```

#### Listing 2: AlturaException.java

```

import java.beans.SimpleBeanInfo;
import java.beans.PropertyDescriptor;
import java.beans.IntrospectionException;

/**
 * Exposes only those {@link AlturaException}'s properties that
 * can be
 * portably serialized according to the SOAP serialization rules.
 */
public class AlturaExceptionBeanInfo extends SimpleBeanInfo {
    public PropertyDescriptor[] getPropertyDescriptors() {

```



---

# can your ecommerce software do this?



## Introducing the pioneer of flexible ecommerce architecture.

Elastic Path is a revolutionary "developer friendly" Java-based ecommerce platform for building sophisticated and evolving online stores. The first to leverage open source frameworks such as Struts and Hibernate, Elastic Path is refreshingly simple to customize, easy to integrate, quick to deploy, and surprisingly cost effective.



To see more visit [www.elasticpath.com/flexible](http://www.elasticpath.com/flexible)

elasticpath

```

PropertyDescriptor pd[] = null;
try {
    pd = new PropertyDescriptor[] { /* key is read-only
*/
        new PropertyDescriptor("key", AlturaException.
            class,
                "getKey", null),
        new PropertyDescriptor("values", AlturaException.
            class,
                "getStringValues", "setStringValues"),
    };
} catch (IntrospectionException e) {
}
return pd;
}
}

```

**Listing 3: TransferObjectList.java**

```

package example;

import java.util.ArrayList;
import java.util.Collection;

public class TransferObjectList extends ArrayList {
    private final CustomerTranferObject EMPTY_ARRAY[] =
        new CustomerTranferObject[0];
    public TransferObjectList() {
    }
    public TransferObjectList(Collection base) {
        super(base);
    }
    public CustomerTranferObject[] getTypedElements() {
        CustomerTranferObject[] array =
            (CustomerTranferObject[]) toArray(EMPTY_ARRAY);
        return array;
    }
    public void setTypedElements(CustomerTranferObject[] elems) {
        clear();
        addAll(java.util.Arrays.asList(elems));
    }
}

```

**Listing 4: TransferObjectListBeanInfo.java**

```

package example;

import java.beans.SimpleBeanInfo;
import java.beans.PropertyDescriptor;
import java.beans.IntrospectionException;

public class TransferObjectListBeanInfo extends SimpleBeanInfo {
    public PropertyDescriptor[] getPropertyDescriptors() {
        PropertyDescriptor pd[] = null;
        try {
            pd = new PropertyDescriptor[] {
                new PropertyDescriptor("typedElements",
                    TransferObjectList.
                        class),
            };
        } catch (IntrospectionException e) {
        }
        return pd;
    }
}

```

**Listing 5**

```

public abstract class PersonTransferObject
    implements java.io.Serializable {
    private static final long serialVersionUID = 1082469451137L;
    private String lastName;
    private String uniqueId;

    public PersonTransferObject(PersonKey key) {
        setKey(key);
    }
    public PersonKey getKey() {
        PersonKey key = new PersonKey();

```

```

        key.setUniqueId(uniqueId);
        return key;
    }
    public void setKey(PersonKey key) {
        this.uniqueId = key.getUniqueId();
    }
    public String getLastName() {
        return lastName;
    }
    public void setLastName(String value) {
        this.lastName = value;
    }

    protected transient Retriever blobRetriever =
        new RetrieverAdaptor();
    public Retriever getBlobRetriever() {
        return blobRetriever;
    }
    public void setBlobRetriever(Retriever blobRetriever) {
        this.blobRetriever = blobRetriever;
    }
}

public class CustomerTranferObject extends PersonTransferObject {
    private static final long serialVersionUID = 10824694501452L;
    private String title;

    public CustomerTranferObject(PersonKey key) {
        super(key);
    }
    public String getTitle() {
        return title;
    }
    public void setTitle(String value) {
        this.title = value;
    }
}

```

**Listing 6: CustomerTransferObjectBeanInfo.java**

```

public class CustomerTranferObjectBeanInfo extends SimpleBeanInfo {
    public PropertyDescriptor[] getPropertyDescriptors() {
        PropertyDescriptor pd[] = null;
        try {
            pd = new PropertyDescriptor[] {
                new PropertyDescriptor("title",
                    CustomerTranferObject.class),
                new PropertyDescriptor("name",
                    CustomerTranferObject.class,
                    "getLastName", "setLastName"),
                new PropertyDescriptor("uniqueId",
                    CustomerTranferObject.class),
            };
        } catch (IntrospectionException e) {
        }
        return pd;
    }
}

```

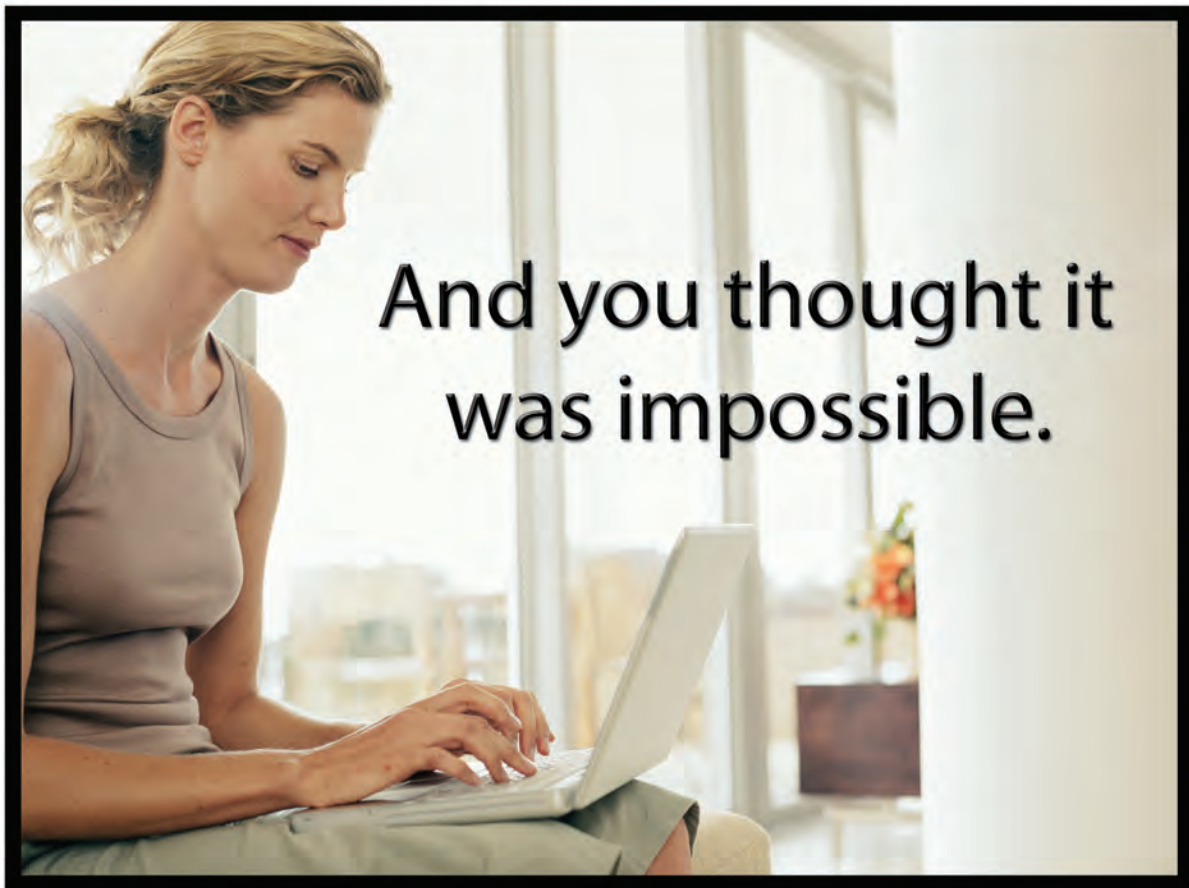
**Listing 7: PersonTransferObjectBeanInfo.java**

```

public class PersonTranferObjectBeanInfo extends SimpleBeanInfo {
    public PropertyDescriptor[] getPropertyDescriptors() {
        PropertyDescriptor pd[] = null;
        try {
            pd = new PropertyDescriptor[] {
                new PropertyDescriptor("name",
                    PersonTranferObject.class,
                    "getLastName", "setLastName"),
                new PropertyDescriptor("uniqueId",
                    PersonTranferObject.class),
            };
        } catch (IntrospectionException e) {
        }
        return pd;
    }
}

```

In five days you'll be  
programming in JSF,  
Spring and Hibernate  
all by yourself.



Public and on-site courses. Online training coming soon.

Phone: 520-290-6588

[info@arc-mind.com](mailto:info@arc-mind.com)

[www.arc-mind.com](http://www.arc-mind.com)



# Seam: The Next Step in the Evolution of Web Applications

*A powerful new application framework for managing contextual components*

by Norman Richards

**W**eb sites were originally static. Later dynamic content came about through CGI scripts paving the way for the first true Web applications. Since HTTP was entirely stateless, it became necessary to invent ways for requests to be linked together in a sequence. At first state was added to the URLs, but later the cookie concept came into being. By giving each user a special token, the server could maintain a context for each user, the HTTP session where the application can store state. As simple as it is, the HTTP session defines the entire concept of what a Web application is today.

The benefits of the HTTP session are clear, but we don't often stop to think about the drawbacks. When we use the HTTP session in a Web application, we store the sum total of the state of the user's interaction with the application in one place. This means that unless we jump through some seriously complex hoops, the user can only interact with an application in one way at a time.

Consider an application that manages a set of paged search results stored in the HTTP session. If the user were to start a new search in a new window, the new search would overwrite any previous session-scoped results. That's painful enough, but multitasking users aren't the only way session-scoped data causes Web applications to explode.

Consider our friend the back button, the mortal enemy of nearly every Web developer. When a user goes backwards in time to a previous page and presses a button, that application invocation expects to be associated with an HTTP session state that may have subsequently changed. When your application only has one context to store state, there's little hope of creating anything but the fragile Web applications that we all struggle with.

Most applications deal with this by adding state to the URLs, either going back to manually putting state information in the URLs or by adding a cookie-like token in the URL to point to a finer-grained context than the HTTP session. This can force the developer to pay a lot of attention to state management, often spending more time on it than on writing the actual application.

## JBoss Seam

This article introduces JBoss Seam, a framework for managing contextual components. You'll see how Seam can manage state information in a Web application, overcom-

ing the fundamental limitations of the HTTP session and enabling entirely new contexts that dramatically extend the capabilities of Web applications. It does all of this while simplifying your Web application development and reducing the amount of code (and XML) you have to write.

Seam is a framework for managing contextual components. What does that mean? Let's first look at a component. They go by many names: JavaBeans, POJOs, Enterprise JavaBeans. They're Java classes that provide some type of function to your application. A Java EE application might have many kinds of components: JSF backing beans creating the Web tier, entity beans providing persistence, and session beans providing business logic. To Seam, they're all components. Seam unifies these diverse component models, letting you think of them all as simply application components.

## A Quick Example

The best way to see what Seam can do is to look at some examples. These examples are derived from the Seam DVD Store application. You can see the complete application in the examples directory of the Seam distribution. We'll start with a simple EJB3 entity bean for a product.

```
@Entity
@Name("product")
public class Product
    implements Serializable
{
    long id;
    String title;
    String description;
    float price;

    @Id @GeneratedValue(strategy=AUTO)
    public long getId() {
        return id;
    }
    public void setId(long id) {
        this.id = id;
    }

    public String getTitle() {
        return title;
    }
}
```

Norman Richards is a JBoss developer living in Austin, Tx. He is co-author of JBoss: A Developer's Notebook and XDoclet in Action.

norman.richards@jboss.com

```

    }

    public void setTitle(String title) {
        this.title = title;
    }

    // more getters and setters.
}

```

This is an EJB3 entity bean, a POJO with a couple of annotations. The only thing that stands out here is the `@Name` annotation, which marks it as a Seam component named `product`. We'll see later how this affects things. For now let's move on to an application component that displays a list of all the products in the system.

```

@Stateful
@Name("search")
@Interceptors({SeamInterceptor.class})
public class SearchAction
    implements Search
{
    @PersistenceContext(type=EXTENDED)
    private EntityManager em;

    @Out
    private List<Product> products;

    @Factory("products")
    public void loadProducts()
    {
        products = em.createQuery("from Product p")
            .getResultList();
    }
}

```

This is a stateful EJB3 session bean. We've marked it as a Seam component using the `@Name` annotation and brought in Seam functionality with the `SeamInterceptor`. Ignoring the remaining Seam annotations for a moment, what we have here is a simple component that keeps track of a list of `Product` objects.

The `loadProducts()` method uses the EJB3 persistence API to load that list of products. It makes use of EJB3 dependency injection to receive an `EntityManager` instance from the container so that it can load the products into the product list.

The goal of this component is to provide this list of products to the UI. The `@Out` annotation on the product list does this. `@Out` marks the list as data to be shared out to the UI.

Let's jump forward to the view. This is part of the Facelets XHTML template, but for our purposes you can think of it as a JSP file that uses the JavaServer Faces tag libraries. The `dataTable` tag renders the list of items in table form using the three column definitions: title, description, and price.

```

<h:dataTable value="#{products}" var="prod">
    <h:column>
        <f:facet name="header">title</f:facet>
        #{prod.title}
    </h:column>
    <h:column>
        <f:facet name="header">description</f:facet>

```

```

        #{prod.description}
    </h:column>
    <h:column>
        <f:facet name="header">price</f:facet>
        #{prod.price}
    </h:column>
</h:dataTable>

```

The search component provides the products value for the table to the view, but how does the value get populated in the view? The `@Factory` annotation on the `loadProducts()` method tells Seam that if a view needs a products value, the `loadProducts()` method can be used as a factory method to load the products from the database.

There's something else happening here. `SearchAction` is a stateful session bean, and the product's value is just part of its state. Stateful components have a lifecycle. They have to be created, kept around, and eventually destroyed. In Seam, stateful components have the lifecycle of their surrounding context. If a stateful component were session-scoped, for example, it would be stored in the HTTP session and destroyed when the HTTP session is destroyed, unless it reaches its natural end of life sooner.

We mentioned earlier that Seam has several more interesting contexts than just simple session-scoped data. Stateful components, by default, have conversational scope. A conversation is a sequence of clicks within a Web application. You can think of it as an actual conversation with part of the application. You interact with one part of the application for a few clicks and say good-bye, maybe moving on to some other part of the application.

To get a better feel for it, let's add some additional functions to the search component and see exactly how the conversation works.

```

static final int PAGE_SIZE = 10;

int    currentPage = 0;
boolean hasMore    = false;

@Begin(join=true)
@Factory("products")
public void loadProducts()
{
    List<Product> items =
        em.createQuery("from Product p")
            .setMaxResults(PAGE_SIZE+1)
            .setFirstResult(PAGE_SIZE*currentPage)
            .getResultList();

    if (items.size() > PAGE_SIZE) {
        products = new ArrayList<Product>(items.subList(0,
                                                    PAGE_SIZE));

        hasMore = true;
    } else {
        products = items;
        hasMore = false;
    }
}

public boolean isFirstPage() {
    return currentPage==0;
}

```

```

public boolean isLastPage() {
    return !hasMore;
}

public String nextPage() {
    if (!isLastPage()) {
        currentPage++;
        loadProducts();
    }
    return null;
}

public String prevPage() {
    if (!isFirstPage()) {
        currentPage--;
        loadProducts();
    }
    return null;
}
}

```

The changes add a little bit of extra state to the search component, namely the concept of a current page in the search results. The `loadProducts()` method now uses the current page number to load only a single page of results. An extra product is loaded as a test to see if there are additional results available in the database. That provides enough information for the `isFirstPage()` and `isLastPage()` methods to determine whether or not the current page is either the first or last page of the results.

Finally, `SearchAction` now provides `nextPage()` and `prevPage()` action methods to alter the current state of the component. These methods work by changing the current page number and reloading the products list. The new products list is pushed out to the view to be displayed.

The listing below shows all the required changes to the view.

```

<h:form>
    <h:dataTable value="#{products}" var="prod">
        <!-- ... same as previous example -->
    </h:dataTable>

    <h:commandButton action="#{search.prevPage}"
        value="previous"
        rendered="#{!search.firstPage}" />

    <h:commandButton action="#{search.nextPage}"
        value="next"
        rendered="#{!search.lastPage}" />
</h:form>

```

The output is wrapped in a form that contains buttons linking to the `prevPage()` and `nextPage()` actions on the search component. Seam components are directly usable from a view by name. Seam will make sure that the appropriate instance for the current context (the current conversation, in this case) is available.

Let's look more closely at this. To display the products list on the page, Seam locates the search component, which the products factory, and loads the data. The `loadProducts()` method is annotated with `@Begin`, which hints to Seam that the current conversation (the one started when the page was first loaded) is a long-running conversation and should be kept around (without this hint, Seam would assume the conversation would expire at the end of the request and remove the conversation, along with any components in that scope).

With a conversation in place, the previous and next buttons automatically link to the action methods on the same instance we're talking to now. As long as the user continues to click through these actions, the conversation is maintained and the users sees the appropriate state.

However, if the user loads the page again or accesses the page in another tab or window, Seam will interpret this as a request for a new conversation and create a new stateful search component to service this conversation. Seam can maintain any number of concurrent conversations with a user, letting the user browse through multiple sets of search results without ever confusing the data between conversations as might happen if the search results were naively put in the HTTP session.

This is an amazingly powerful concept considering the component didn't have to be specially coded to be able to do it. The search component simply implements the logic required to maintain its own state. Seam takes care of controlling the visibility and lifecycle of the instances.

To complete the example and show a few extra features of Seam, we'll add the ability to select specific products and add them to a shopping cart. We'll start with a basic shopping cart component.

```

@Stateful
@Name("cart")
@Scope(ScopeType.SESSION)
@Interceptors(SeamInterceptor.class)
public class ShoppingCartBean
    implements ShoppingCart
{
    private List<Product> contents =
        new ArrayList<Product>();

    public List<Product> getContents() {
        return contents;
    }

    public void addToCart(Product product) {
        contents.add(product);
    }

    @Destroy @Remove
    public void destroy() {
    }
}

```

“Seam overcomes the fundamental limitations of the HTTP session and dramatically extends the capabilities of Web applications”



It's another stateful session bean. The state of the cart is the list of products in the cart, and the bean provides a method for adding a product to the cart. Just like the search component, the cart is a simple POJO concerned only about managing its state and providing useful functions to the application.

The shopping cart is a session-scoped bean. No matter how many simultaneous searches a user does, items should be added to the same cart instance. Session scope matches this perfectly, so we use the `@Scope` annotation to override Seam's default conversational scope for stateful components.

We're done with the shopping cart, so now we'll head back to the search component and its UI view. We need a simple way to operate on items in the products list. JSF provides a `DataModel` interface that can be used in conjunction with the `dataTable` tag to provide a concept of selecting an individual row of the table. Using it, however, would couple our application to the UI and move the search component more towards being a UI component than an application component.

Seam provides the solution here by providing UI wrapper annotations that let you construct UI components out of internal state. By replacing the `@Out` tag with the `@DataModel` tag on the products list, Seam will make sure that the UI sees a proper JSF `DataModel` instead of a simple list.

```
@DataModel
private List<Product> products;
```

A `DataModel` can have a selected item, and Seam can inject the selected item back into the search component using the `@DataModelSelection` annotation. Whenever an action is done on a row in the products `dataTable`, the `selectedProduct` value will be set accordingly.

```
@DataModelSelection
Product selectedProduct;
```

The `dataTable` can be easily updated with a button to add a specific item to the cart. The following code adds a "Buy it!" button on each row of the table.

```
<h:dataTable value="#{products}" var="prod">
  <!-- columns for title, description, and price -->
  <h:column>
    <f:facet name="header">Add to Cart</f:facet>
    <h:commandButton action="#{search.select}"
      value="Buy it!" />
  </h:column>
</h:dataTable>
```

The button maps to the `select` action on the search component. (Which one? The one in the current conversation.) Seam will set the `selectedProduct` field, so the `select()` method implementation is easy, if we have a reference to correct the `ShoppingCart` instance.

```
public String select() {
    cart.addToCart(selectedProduct);
    return null;
}
```

```
}
```

To get a reference to the cart, we can ask Seam to inject it using the `@In` annotation.

```
@In(create=true)
ShoppingCart cart;
```

The default scope for the shopping cart is session, so Seam will try to find the cart instance in the session. If there isn't a current instance, the `create=true` parameter asks Seam to create an instance in the session for future use.

Seam can manage injection and outjection between components in differing contexts. In this case, a conversational component has injected a component from a larger context. However, we could have easily worked the other way. The `buy-it` action could easily have been written to act on the cart component.

```
<h:commandButton action="#{cart.select}"
  value="Buy it!" />
```

In that case, the shopping cart could have injected the search component and retrieved the current object from it. Assuming the search component has a `getSelectedProduct()` method that returns the selected product, it would look like this:

```
@In(required=false)
Search search;

public String select() {
    addToCart(search.getSelectedProduct());
    return null;
}
```

This is a kind of injection that other dependency injection systems just can't handle. The shopping cart, a stateful bean, can ask for the search component from a narrower context that might change from request to request. (Remember, there's one cart servicing multiple conversation-scoped searches.) Seam ensures that any time the shopping cart is used, it will have a reference to the search component for the current context. Pushing products from the search action to the cart seems more intuitive in this case, but it's nice to know that Seam provides powerful enough mechanisms to allow the components to be written in whatever manner is most intuitive for the domain.

## Business Process

Seam shines at managing component links across varying contexts. We've introduced the conversation context, allowing for components that are finer-grained than the HTTP session. As a last example, we'll see how Seam can manage contextual data that's shared between users.

It's not that hard to share state between users in a Web application. One option for sharing state between multiple users in a single server is the application context. However, just like the HTTP session is too broad a context for component interaction for a single user, the application scope is also too broad for simple state communication between users.

## “Seam applications and application components can be automatically generated, so a Seam-based application can get up and running quickly”

A more refined mechanism for sharing state information between multiple users is by using a business process. Think of a business process as a flow of conversations between users. A customer has a conversation with a search component that culminates in the creation of an order. Later a store employee might engage in another conversation to review or ship the order. The shared context throughout this is the order itself. Seam lets us code components to do the review and shipping acts that are as simple as the search and cart components we've already seen.

Let's take a very simple order fulfillment process written using the jBPM process definition language. This simple process has one state with a ship task that has to be acted on by the store manager. Once the task is completed the process is done.

```
<process-definition name="OrderManagement">
  <start-state>
    <transition to="process"/>
  </start-state>

  <task-node name="process">
    <task name="ship">
      <assignment actor-id="manager"/>
    </task>
    <transition name="shipped" to="complete" />
  </task-node>

  <end-state name="complete"/>
</process-definition>
```

This is a very trivial process, but it does illustrate the basics of process integration. Once you see how it works, it won't be hard to imagine how it can be extended to more complex processes later.

Let's suppose that we've added a purchase action to the shopping cart that let the user complete the checkout. We'd like to have an instance of the business process created afterwards. With Seam, this is as easy as annotating the method with @CreateProcess.

```
@Out(scope=BUSINESS_PROCESS, required=false)
long orderId;
@Out(scope=BUSINESS_PROCESS, required=false)
float amount;
@Out(scope=BUSINESS_PROCESS, required=false)
String email;

@CreateProcess(definition="OrderManagement")
public String purchase() {
    // process the order
    // set the orderId, amount and email fields
}
```

The @Out annotations here are on the state of the current object that we want to be shared out with the business process context. By associating the orderId with process, any component that's taking part in the process can inject the orderId and use the data. The component taking part in the action doesn't have to know where the state is coming from. It only has to know that it will get the state it needs to do the task required of it.

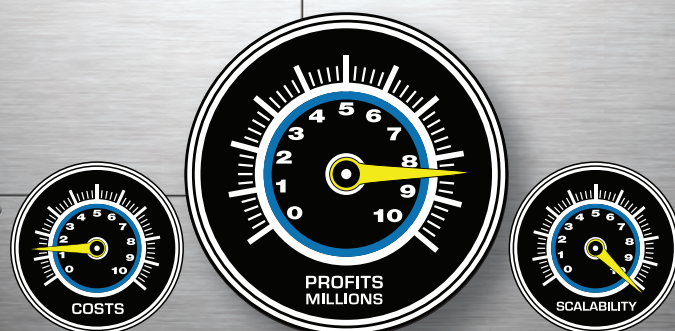
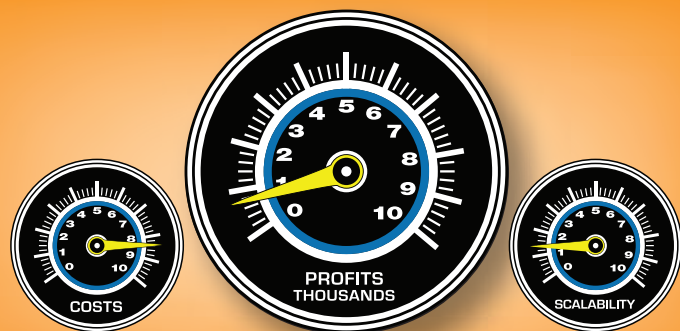
When the process instance is created, a ship task will be created and assigned to the manager user. Seam provides several built-in components for querying tasks. The taskInstanceListForType component provides a list of a tasks by type for the current user. The following code displays a list of ship tasks for the user:

```
<h:dataTable value="#{taskInstanceListForType['ship']}"
  var="task">
  <h:column>
    <f:facet name="header">Order Id</f:facet>
    #{task.variables['orderId']}
  </h:column>
  <h:column>
    <f:facet name="header">Order Amount</f:facet>
    <h:outputText value="#{task.variables['amount']}">
      <f:convertNumber type="currency"
        currencySymbol="$" />
    </h:outputText>
  </h:column>
  <h:column>
    <h:column>
      <h:commandLink action="#{ship.ship}">
        <f:param name="taskId" value="#{task.id}"/>
      </h:commandLink>
    </h:column>
  </h:column>
</h:dataTable>
```

The order ID and amount are displayed, followed by a button to ship the item. In the real application clicking on the button engages the store manager in a conversation with a shipping component where the manager can examine the order and either enter a tracking number or cancel the order. In this example, we'll use a much simpler component that simply changes the state of the order object.

```
@Stateful
@Name("ship")
@Interceptors(SeamInterceptor.class)
public class ShipAction
  implements Ship,
    Serializable
{
```

# A slapdash approach won't get you there...



## ExtenXLS will.

**NEW!**  
Round Trip  
Reporting™



"ExtenXLS makes it possible to re-use spreadsheets representing a tremendous investment in time and business logic and to 'connect' those spreadsheets to various SQL data sources to provide access to usable data in a way never before possible."

- Project Leader, AT&T

"We use ExtenXLS for 'pushing' accounting data into an Excel template. A key customer wanted web access to Excel format reports. We tried .NET but had problems with preserving charts & formatting. ExtenXLS provides a flexible, cost effective solution that gives us the ability to stay ahead of user requirements."

- Project Leader, John J. McMullen Company

### It's What's behind the Dashboard that Counts!

Slapping a dashboard onto a static spreadsheet file may **look** good, but it's like installing a flashy dashboard on an outdated clunker. Install that same front-end on ExtenXLS — a **scalable, server-based spreadsheet engine** — and watch your flashy dashboard come alive.

### Now with Round Trip Reporting™

Reuse your existing spreadsheets as a data-entry tool. Modified data is extracted from your spreadsheets and turned into Java Data Objects for reuse in all your programs.

### Reach New Heights without the Learning Curve

With the familiar look and feel of Excel™, ExtenXLS eliminates the learning curve imposed on your end-users by other reporting tools. ExtenXLS unlocks the business logic stored in static spreadsheets throughout your organization, saving time and money.

### Escape the Gravitational Pull of Obsolete Reporting!

Output to customized HTML for maximum compatibility or to XML for further processing. Keep your users happy with native Excel™ output which preserves the VB macros, images, charts, and other features that transform live data into actionable reports. You can even embed a familiar spreadsheet component in your Swing applications.

**Don't rely on a slapdash approach for your mission-critical reporting needs.  
Put a rocket under the hood and achieve escape velocity.**

Visit the mother ship at: [www.extentech.com/jdj](http://www.extentech.com/jdj) and take your free evaluation copy into orbit for a test flight.

**EXTENXLS4™**  
JAVA XLS REPORTING TOOLKIT

Java is a Registered Trademark of Sun Microsystems Inc. Excel is a Registered Trademark of Microsoft Corporation. All other trademarks mentioned herein are the property of their respective owners.

**extentech™**  
Call Us: 415-759-5292



```

@PersistenceContext(type=EXTENDED)
EntityManager em;

@In
Long orderId;

@BeginTask
@EndTask
public String ship() {
    Order order = (Order) em.find(Order.class,
                                   orderId);

    order.ship();

    return "admin";
}
}

```

The `@BeginTask` annotation takes the task associated with the `taskId` parameter and makes the corresponding process state available to the component. That allows the `orderId` to be injected and the corresponding order shipped. Since we've greatly simplified the flow here, this method uses the `@EndTask` annotation to signal the completion of the task.

If the operation succeeds, the process will advance. In this simple example, the process completes, but if the process were more complex, other tasks, possibly for other users, might be created. Those new tasks would see any changes to the process state.

We won't add any more tasks here. Instead let's make one tiny change to the process to illustrate how business process can further simplify the flow of the application. Let's imagine that we wanted to send an e-mail after shipping the order. We could do that in the `ship()` action, but then we'd be mixing the concerns of components. The `ship` component should only be concerned with the single action of shipping the order.

There are many ways to add actions to a process in jBPM. We'll associate the e-mail action with the transition out of the state.

```

<transition name="shipped" to="complete">
    <action expression="#{shippingEmail.send}" />
</transition>

```

This expression invokes the `send` action on the `shippingEmail` component. It's easy to see how a Seam component could inject state from the process, pull out the information to put in the e-mail, and use `JavaMail` to notify the user that the order has shipped.

Using a process lets us separate out process concerns like sending a notification from application concerns like marking an order as shipped in the database. Seam lets us integrate with the business process without coupling the code to business process APIs or complicating the components with complex state management. Seam simplifies the application by letting components focus only on the exact task they're meant for and delegating all glue code to Seam.

## One More Thing...

There's a lot more to say about Seam. This article has focused entirely on the management of contextual components to provide for simpler Web application development. However, Seam goes well beyond that. Let's look at some of the other things that make Seam interesting.

The conversation model in Seam is much richer than what is shown here. Seam can carry on conversations with multiple components on a single page and switch between conversations. It has additional contexts beyond the ones mentioned so far. One example is the page context that stores state from one request for a subsequent request.

Seam provides for model-based entity validation. Most Web application require user input to be validated. Sometimes this is done on the client side and sometimes it's done on the server side, but it's almost always a function of the UI. Seam provides an additional layer of model validation. Instead of only being about to say that an input field has to contain a valid e-mail address, an entity could use the Hibernate validation annotations to express that the e-mail address of the person entity should be a valid e-mail address. Seam can apply these validations any time an entity is used in a form.

Seam supports the use of jBPM definition to describe the flow of pages in a conversation. jBPM pageflow is simpler and cleaner than JSF navigation, and allows for a separation of flows in an application. Pageflow definitions can be created and maintained using the visual process editor in jBPM.

Seam applications and application components can be automatically generated, so a Seam-based application can get up and running quickly. This includes the now-popular generation of CRUD applications from a database schema.

Testing is central to JBoss Seam. Seam supports testing user interactions with components in unit tests. These tests can be run, either in the IDE or from your build script, without involving an application server.

Finally, JBoss Seam supports non-EJB development. This article shows EJB3 integration, but Seam also supports Hibernate persistence directly. Application logic can be written using plain `JavaBeans` instead of EJB3 session beans. Seam applications can run inside any application server or even in a bare Web container like Tomcat.

## Conclusion

Seam is a powerful new application framework for managing contextual components. Seam represents the next logical step in the evolution of applications, letting Web application components take advantage of a much richer set of contexts without polluting the components with code unrelated to the core function of the component. Seam is fast and easy and represents the state-of-the-art of modern Web development. ☺

## Resource

- <http://www.jboss.com/products/seam>

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

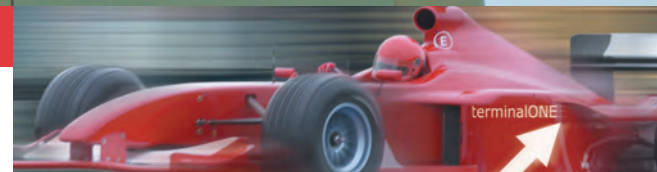


**The future of EDI is B2B over IP**

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](http://www.xenos.com/VAN)**

1 888 242 0695

1 905 763 4468

[terminalONE@xenos.com](mailto:terminalONE@xenos.com)



# What Is POJO Programming?

Introducing POJO application development

by Chris Richardson

**T**he novel *A Deepness in the Sky* by Vernor Vinge is set in the distant future. The character Pham Nuwen is responsible for maintaining software whose components are thousands of years old. Today, however, it's difficult to imagine maintaining an Enterprise Java application for more than a few years. More often than not, the application is tightly coupled to infrastructure frameworks that evolve rapidly in ways that don't preserve backwards compatibility. Consequently, upgrading to a new and improved framework can be challenging and risky.

Fortunately, there's now a much better way to build Enterprise Java applications: Plain Old Java Objects (POJOs), which are classes that don't implement infrastructure framework-specific interfaces, and non-invasive frameworks such as Spring, Hibernate, JDO, and EJB 3, which provide services for POJOs.

Using POJOs future proofs your application's business logic by decoupling it from volatile, constantly evolving infrastructure frameworks. Upgrading to a new version or switching to a different framework becomes easier and less risky. POJOs also make testing easier, which simplifies and accelerates development. Your business logic will be clearer and simpler because it won't be tangled with the infrastructure code. And, as an added bonus, you can often deploy a POJO application using a simpler, Web container-only application server.

In this article, you'll learn about POJO programming. I'll begin by describing the concept of POJOs. You'll then get an overview of persisting POJOs with Hibernate and EJB 3 and making them transactional with EJB 3 and Spring. Of course, it's unlikely that the applications we're developing today will be used thousands of years from now but until their demise, using POJOs will make it easier to upgrade them to use newer frameworks.

## What Is a POJO?

Even though it has tremendous benefits, the concept of a POJO is remarkably simple. A POJO is a Java object that doesn't implement any special interfaces such as those defined by the EJB 2 framework. Martin Fowler, Rebecca Parsons, and Josh MacKenzie coined the name to give regular Java objects an exciting-sounding name that encouraged developers to use them ([www.martinfowler.com/bliki/POJO.html](http://www.martinfowler.com/bliki/POJO.html)).

To contrast the EJB and POJO approaches consider the following example. Imagine that you worked for a bank and needed to implement a service to transfer money from one bank account to another. If you were using EJB2, your code would most likely consist of a `MoneyTransferService` stateless session bean that coordinated the transfer of money between the accounts and an `Account` entity bean that accessed the account data. The problem with this design is that each EJB would be a mixture of business logic and EJB 2 infrastructure code. They would be intimately coupled to the EJB 2 framework. Furthermore, they would be difficult to test without deploying unless you jumped through the kinds of hoops described in `TwoLevelDomain` (<http://www.theserverside.com/articles/article.tss?!=TwoLevelDomainModel>).

By comparison, the POJO version (which is downloadable from [www.pojosinaction.com](http://www.pojosinaction.com)) would look something like Listing 1. The `MoneyTransferService` and its implementation class define a `transfer()` method and the `Account` class maintains a balance and defines `debit()` and `credit()` methods. The `AccountDAO` is the interface for the Data Access Object (DAO) class whose implementation I'll describe later.

As you can see, these are regular Java classes. None of them implement special interfaces or call any framework classes. What's more, the `MoneyTransferService` doesn't instantiate the `AccountDAO` directly or look it up using an API such as JNDI. Instead, the `AccountDAO` is passed as a constructor parameter. This is an example of what is termed dependency injection, which is one of the key enablers for POJO development (<http://www.martinfowler.com/articles/injection.html>). In this example, dependency injection decouples the `MoneyTransferService` from the infrastructure framework. The `MoneyTransferService` only depends on the `AccountDAO` interface – not on the framework used to implement the `AccountDAO` or an API such as JNDI.

You could, of course, simplify the `MoneyTransferService` by dispensing with the `AccountDAO` and directly injecting, for example, a `Spring HibernateTemplate` or an `EJB 3 EntityManager`. However, I generally prefer to keep my framework-specific data access code separate from my business logic.

## The Benefits of POJOs

Decoupling the application code from the infrastructure frameworks is one of the many benefits of using POJOs. They

**Chris Richardson** is the author of the recently published *POJOs in Action*. He's a developer, architect, and mentor with over 20 years of experience. Chris runs a consulting company that jumpstarts new development projects and helps teams that are frustrated with Enterprise Java to become more productive and successful. He lives in Oakland, CA with his wife and three children.

[cer@acm.org](mailto:cer@acm.org)



# 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](http://www.xythos.com/neu)*

**For more information please call 1.888.4XYTHOS or visit [www.xythos.com](http://www.xythos.com)**  
Xythos Software, Inc., 655 Montgomery, Suite 1600, San Francisco, CA 94111

**Xythos**  
software, inc.

also simplify development because rather than being forced to think about everything — business logic, persistence, transactions, etc. — at once, you can focus instead on one thing at a time. You can design and implement the business logic and then, once that's working, you can deal with persistence and transactions.

POJOs also accelerate development. You can test your business logic outside of the application server and without a database. You don't have to package your code and deploy it in the application server. You also don't have to keep the database schema constantly in sync with the object model or spend time waiting for slow-running database tests to finish. Tests run in a few seconds and development can happen at the speed of thought — or at least as fast as you can type!

But by now you probably have questions such as what about transactions? Security? Persistence? Remoting? And how do you assemble an application with dependency injection? As you might have guessed, POJOs by themselves are insufficient. To use them in an enterprise application you have to use one or more frameworks.

### Overview of Frameworks for POJOs

When developing an enterprise application, you need services such as transaction management, security, and persistence. You also need a way to assemble components together and access enterprise resources such as JDBC DataSources. One option is to use some very popular non-EJB frameworks such as Hibernate, JDO, and Spring. The Hibernate and JDO frameworks provide persistence for POJOs. The Spring framework provides services for POJOs such as transaction management and dependency injection. It also has support for POJO remoting and is the foundation of the Acegi Security framework (<http://acegisecurity.sourceforge.net>) that provides security for POJOs. One big advantage of not using the EJB container is that you can sometimes deploy your POJO application in a simple (and sometimes cheaper) Web container-only server.

Another option is to use the emerging EJB 3 standard, which has adopted many POJO concepts and is a big improvement over EJB 2. In EJB 3, EJBs are very POJO-like in not implementing any special interfaces. Furthermore, EJB 3 entity beans are intended to be the standard persistence mechanism for Java and work both inside and outside the EJB container.

One important benefit of EJB 3, Spring, Hibernate, and JDO is that they are non-invasive. Unlike older technologies such as EJB2, they provide services such as transactions and persistence without requiring the application classes to implement any special interfaces. Moreover, only a small fraction of your code must call any APIs. Classes can be transactional or persistent while still being POJOs.

To use the services provided by these frameworks you must write metadata that configures the frameworks. The metadata comes in the form of either Java 5 annotations or XML configuration files. Later in this article, I'll provide examples of how you use this metadata to map POJOs to a relational database and make them transactions.

Now that you have had an overview of POJOs and their associated frameworks, let's look at dependency injection, persistence, and transaction management in more detail.

### Injecting Dependencies into POJOs

Earlier we saw how the AccountDAO was passed as a constructor parameter to the MoneyTransferService. This kind of dependency injection is called *constructor injection*. Another kind of dependency injection is *setter injection* and consists of passing dependencies as setter parameters. The third kind of dependency injection is *field injection* and involves initializing fields directly. I prefer to use constructor injection because it ensures that an object is constructed with the required dependencies but setter and field injection are also useful.

Dependency injection has the following benefits:

- *Simpler code* - It eliminates the need to call lookup APIs. Because a component's dependencies are passed to it you no longer have to write tedious JNDI code to look up a dependency.
- *Decouples application components from one another and the infrastructure framework* - Dependency injection lets you construct an application from loosely coupled components. Components depend mainly on interfaces rather than on concrete implementations. There are no calls to framework-specific lookup code. The dependencies on the infrastructure frameworks are localized to only those components that call them directly such as DAO implementations.
- *Easier testing* - You can test components in isolation by injecting mock implementations. For example, we can write tests for the MoneyTransferService that use a mock implementation of the AccountDAO. These kinds of tests are essential if you want to be able to work on your business logic without worrying about persistence. They also run a lot faster than those that access a database.

There has to be some application start-up code that instantiates the components and wires them together. You could write the start-up code yourself. For example, you could write some code that creates an instance of a class that implements AccountDAO and then passes it to an instance of the MoneyTransferServiceImpl. However, in a typical application it's usually more convenient to let Spring or EJB 3 inject the dependencies for you.

### Spring Dependency Injection

Dependency injection is one of the core features of the Spring framework. To use it, you must configure Spring's bean factory, which is a sophisticated factory that instantiates objects and wires them together using either constructor injection or setter injection. Here's an example of how to do that:

```
<bean name="/moneyTransferService"
      class="... MoneyTransferServiceImpl">
  <constructor-arg ref="/accountDAO"/>
</bean>

<bean name="/accountDAO"
      class="... AccountDAOHibernateImpl">
  ...
</bean>
```



# Welcome to the Future of Video on the Web!

**REGISTER NOW!**  
[www.iTVcon.com](http://www.iTVcon.com)

**CALL FOR PAPERS NOW OPEN!**

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

**iTV CON.COM**  
INTERNET TV CONFERENCE & EXPO 2006

**Coming in 2006 to New York City!**

“Internet TV is wide open, it is global, and in true ‘Web 2.0’ spirit it is a direct-to-consumer opportunity!”



**For More Information, Call 201-802-3023  
or Email [itvcon@sys-con.com](mailto:itvcon@sys-con.com)**

## Welcome to the Future!

Did you already purchase your “.tv” domain name?

**You can’t afford not to add Internet TV to your Website in 2006!**

2005 was the year of streaming video and the birth of **Internet TV**, the long-awaited convergence of television and the Internet. Now that broadband is available to more than 100 million households worldwide, every corporate Website and every media company must now provide video content to remain competitive, not to mention live and interactive video Webinars and on-demand Webcasts.

20 years ago the advent of desktop publishing tools opened the doors for the creation of some of today’s well-known traditional print media companies as well as revolutionized corporate print communications. Today, with maturing digital video production, the advent of fully featured PVRs, and significant advances in streaming video technologies, **Internet TV** is here to stay and grow and will be a critical part of every Website and every business in the years to come.

It will also very rapidly become a huge challenge to network and cable television stations: **Internet TV** is about to change forever the \$300BN television industry, too.

The Internet killed most of print media (even though many publishers don’t realize it yet), Google killed traditional advertising models, and **Internet TV** will revolutionize television the way we watch it today. You need to be part of this change!

**Jeremy Geelan**  
Conference Chair, iTVCon.com  
[jeremy@sys-con.com](mailto:jeremy@sys-con.com)

PRODUCED BY  
**SYS-CON**  
EVENTS

### List of Topics:

- > Advertising Models for Video-on-demand (VOD)
- > Internet TV Commercials
- > Mastering Adobe Flash Video
- > How to Harness Open Media Formats (DVB, etc)
- > Multicasting
- > Extending Internet TV to Windows CE-based Devices
- > Live Polling During Webcasts
- > Video Press Releases
- > Pay-Per-View
- > Screencasting
- > Video Search & Search Optimization
- > Syndication of Video Assets
- > V-Blogs & Videoblogging
- > Choosing Your PVR
- > Product Placement in Video Content
- > UK Perspective: BBC’s “Dirac Project”
- > Case Study: SuperSun, Hong Kong

- |                |  |
|----------------|--|
| <b>Track 1</b> | Corporate marketing, advertising, product and brand managers   |
| <b>Track 2</b> | Software programmers, developers, Website owners and operators   |
| <b>Track 3</b> | Advertising agencies, advertisers and video content producers  |
| <b>Track 4</b> | Print and online content providers, representatives from traditional media companies, print and online magazine and newspaper publishers, network and cable television business managers |



Each Spring bean definition includes a name, an implementation class, and one or more dependencies, which are supplied by either constructor or setter injection. When the presentation tier asks the bean factory to create a `MoneyTransferService`, first it will instantiate an `AccountDAOHibernateImpl`, which is the Hibernate implementation of the `AccountDAO` interface. The bean factory will then instantiate the `MoneyTransferServiceImpl` passing the `AccountDAOHibernateImpl` to its constructor.

Spring provides an extremely flexible dependency injection mechanism. It can inject arbitrary POJOs in POJOs. Spring can even, for example, do a JNDI lookup to get an object and inject it into a POJO.

### EJB 3 Dependency Injection

In comparison, EJB 3 dependency injection is limited to injecting values from JNDI into either session beans or message-driven beans. You can configure field or setter injection using either XML or annotations. Listing 2 shows how you could inject an `AccountDAO` into a `MoneyTransferServiceImpl` using an annotation.

In this example, the `@EJB` annotation specifies that when the EJB 3 container instantiates a `MoneyTransferServiceImpl` it must initialize the `accountDAO` field with a reference to an `AccountDAO`. Behind the scenes, the EJB 3 container does a JNDI lookup of the `AccountDAO`, which is also implemented as a session bean. If the `MoneyTransferServiceImpl` called the EJB 3 persistence APIs directly then you'd simply inject the EJB 3 `EntityManager` directly.

Dependency injection is one of the cornerstones of POJO development. Another one is the ability to persist POJOs with an object/relational mapping (ORM) framework.

### Persisting POJOs

POJOs often have to be persisted in a relational database. For example, the `Money Transfer` application must store `Account` objects in the `ACCOUNT` table. It could, of course, access the database using JDBC or iBATIS. Sometimes this is the right approach. But unless you have a really simple object model, it can be a lot of work to develop and maintain the Data Access Objects

(DAOs) and the SQL needed to transfer objects to and from the database. It's usually a lot easier to use an ORM framework.

### An Overview of ORM

When using an ORM framework you declaratively specify how your object model maps to the database schema. You specify how classes map to tables; fields map to columns, and relationships map to either foreign keys or join tables. The ORM framework then generates the SQL statements to create, find, update, and delete persistent objects. It also keeps track of which objects have changed and automatically writes them back to the database. As a result, you have to write a lot less code. Moreover, it's usually a lot easier to switch databases because the ORM framework is generating the SQL for you.

EJB 2 CMP provides a primitive form of ORM. However, it requires your classes to implement or extend EJB interfaces. In comparison, modern ORM frameworks such as Hibernate, JDO, and EJB 3 provide what's termed transparent persistence. Apart from perhaps having to add a field to store the primary key, the code for the objects shows no sign that they are persistent. The only classes that are dependent on the ORM framework are classes such as DAOs that create, find, and delete persistent objects. Let's look at an example.

### Declarative Mapping

When using an ORM framework you define the mapping using either XML or Java 5 annotations. Table 1 shows how you map the `Account` class to the `ACCOUNT` table using EJB 3 annotations and a Hibernate 3 XML mapping document.

In each example, the `Account` class is mapped to the `ACCOUNT` table; the `id` field that stores the primary key is mapped to the `ACCOUNT_ID` column; and the `balance` field is mapped to the `BALANCE` column.

### AccountDAOHibernateImpl

Once you've defined the mapping you can use the ORM framework's API for creating, finding, and deleting persistent objects. The APIs provided by each of the ORM frameworks are pretty similar. Here's an example of a DAO that uses a Spring `HibernateTemplate`, which is an ease-of-use wrapper around the Hibernate APIs:

EJB 3	Hibernate 3
<code>@Entity</code>	<code>&lt;class name="Account"</code>
<code>@Table(name="ACCOUNT")</code>	<code>table="ACCOUNT"</code>
<code>class Account {</code>	<code>&lt;id name="id"</code>
<code>...</code>	<code>column="ACCOUNT_ID"&gt;</code>
<code>@Id</code>	<code>&lt;generator</code>
<code>@Column(name="ACCOUNT_ID")</code>	<code>class="native" /&gt;</code>
<code>private int id;</code>	<code>&lt;/id&gt;</code>
<code>@Column(name="BALANCE")</code>	<code>&lt;property name="balance"</code>
<code>private double balance;</code>	<code>column="BALANCE"/&gt;</code>
<code>...</code>	<code>...</code>
<code>}</code>	<code>&lt;/class&gt;</code>

Table 1

```
public class AccountDAOHibernateImpl
    implements AccountDAO {
    private HibernateTemplate hibernateTemplate;
    public AccountDAOHibernateImpl(HibernateTemplate
                                    template) {
        hibernateTemplate = template;
    }
    public Account findAccount(String accountId) {
        return (Account)
            hibernateTemplate.get(new Integer(accountId));
    }
    ...
}
```



**Conferences:** April 3 – 6, 2006

**Expo:** April 4 – 6, 2006

Boston Convention  
& Exposition Center

**2 Conferences**

**10 Tracks**

**100+ Sessions**

**Over 200 Exhibitors**

Introducing: **OpenSolutions**  
WORLD™

OPEN Source.  
OPEN Solutions.



OPEN. For Business.

#### LinuxWorld Conference tracks

- Mobile & Embedded Linux
- Network Management
- Kernel and Driver Development
- Linux on the Desktop

#### OpenSolutions World Conference tracks

- Business Case
- Security
- Open Source Applications
- Enterprise Application Development
- Dynamic IT: Cluster/Grid/Virtualization
- Emerging Trends

#### Open source means business.

Find out what it can mean for your business at LinuxWorld Conference & Expo—in ten tracks and more than 100 in-depth sessions, tutorials, and workshops led by the open source movement's top minds.

#### Go beyond Linux.

Expand your vision at OpenSolutions World, a new conference at LinuxWorld covering the full spectrum of open source solutions and strategies for the enterprise.

#### Get the full picture.

Explore a comprehensive exhibition of open source products, technologies, and services from more than 200 key Linux and open source vendors—all under one roof.

#### Put it to work.

Come to LinuxWorld Conference & Expo April 3 – 6, 2006, and harness the power of the entire open source community for your business. The future of enterprise technology is wide open.

**Register by 3/3 and save**

Enter priority code D0303 and save up to \$500

[www.linuxworldexpo.com/boston](http://www.linuxworldexpo.com/boston)



© 2005 IDG World Expo Corp. All rights reserved. LinuxWorld Conference & Expo, LinuxWorld and .org Pavilion are registered trademarks of International Data Group, Inc. All other trademarks are property of their respective owners.

D0303

The `HibernateTemplate` is passed to the `AccountDAOHibernateImpl` using constructor injection. The `findAccount()` method loads an account by calling `HibernateTemplate.get()`. When `get()` is called, Hibernate generates and executes a `SELECT` to load the corresponding row from the `ACCOUNT` table. If the `Account` object is then modified by the application, Hibernate will execute an `UPDATE` statement to save the changes in the database. An EJB 3 DAO would work the same way. It would call the EJB 3 `EntityManager` to create, find, and delete persistent objects.

### But There Are Also Some Drawbacks and Limitations

When used appropriately ORM is an extremely powerful tool that can significantly increase development productivity while providing excellent performance. But like every tool, it has its limitations. For example, an ORM framework might not be a good choice in the following situations:

- Your application accesses a denormalized legacy schema
- Your DBA insists on inspecting your SQL statements
- Your DBA requires all database accesses to go through stored procedures.
- Your application must perform bulk updates or inserts.

Don't be afraid to use SQL when you have to.

### Making POJOs Transactional

Transactions are an essential part of the Enterprise Java application. Without them we risk corrupting data. Consider, for example, the `MoneyTransferService` shown earlier in Listing 2. It's vital that when the `transfer()` method executes in a transaction that you ensure that updates are made atomically. Otherwise, if the application crashes after debiting one account but before crediting the other account the money could disappear into the ether.

One of the nice features of EJB2 session beans is its support for container-managed transactions, which are a form of declarative transaction management. They simplify the application and make it more reliable by eliminating error-prone transaction management code. In a POJO application, the two main ways to implement declarative transaction management are to use either the Spring framework or EJB 3. Both frameworks use either Java 5 annotations or XML to specify the transactional attributes of a POJO.

### Spring Transaction Management

The Spring bean factory does more than simply instantiate objects. It can also wrap the objects that it creates with interceptors (a.k.a. proxies). These interceptors are how Spring provides a simple, yet effective Aspect-Oriented programming (AOP) implementation. AOP is the foundation of Spring transaction management. To make a POJO transactional you configure the bean factory to wrap it with `TransactionInterceptor`, which executes method calls within a transaction.

Spring provides a couple of ways to configure the bean factory to apply the `TransactionInterceptor`. One option is to use the `@Transactional` annotation on the interface, implementation class, or individual methods. For example:

```
@Transactional
```

```
interface MoneyTransferService {..}
```

The `@Transactional` specifies that when the Spring bean factory instantiates the `MoneyTransferServiceImpl` that implements the interface, it must apply `TransactionInterceptor`.

Another option is to write XML bean definitions that explicitly apply the `TransactionInterceptor` to the POJO. Here's an example bean definition that uses the Spring `BeanNameAutoProxyCreator` to apply a `TransactionInterceptor` to the `MoneyTransferService`.

```
<bean id="transactionProxyCreator" class="org.springframework.aop.framework.autoproxy.BeanNameAutoProxyCreator">
  <property name="beanNames">
    <list>
      <idref bean="moneyTransferService" />
    </list>
  </property>
  <property name="interceptorNames">
    <list>
      <idref bean="transactionInterceptor" />
    </list>
  </property>
</bean>
```

The XML is more verbose than the annotation but has the advantage of leaving the source code unchanged. It also works with older JDKs that don't support annotations.

Spring doesn't implement transactions itself but is, instead, a wrapper around other transaction management APIs. Unlike EJB 3, it gives you the flexibility of using either JTA (as provided by an application server) or the transaction management APIs provided by ORM frameworks such as Hibernate and JDO.

### EJB 3 Transaction Management

In EJB 3, declarative transaction management is, as you might expect, one of the built-in features of session and message-driven beans. You configure transaction management using either Java 5 annotations or a more traditional EJB deployment descriptor. This means, for example, that the `MoneyTransferService` EJB shown in Listing 2 will automatically be transactional.

Now that we've looked at how to make POJOs transactional and persistent let's examine the dangers of using annotated POJOs.

### Beware of Annotated POJOs

The examples in this article use both XML and annotations to configure POJOs. However, the trend in POJO programming is away from XML configuration files and towards annotations. Annotations are certainly convenient because they're physically next to the program element (class, field, method) that they configure. The metadata is more concise and easier to maintain. There's certainly nothing wrong with using application-specific annotations. But, the problem with framework-specific annotations is that they couple your application to the framework in the same way that an API call does ([http://www.aspectprogrammer.org/blogs/adrian/2004/08/when\\_is\\_a\\_pojo.html](http://www.aspectprogrammer.org/blogs/adrian/2004/08/when_is_a_pojo.html)).



For example, if you use Spring's `@Transactional` attribute then your classes won't compile without Spring. Similarly, if you use EJB 3 annotations then your classes will be coupled to the EJB 3 framework. These annotations won't stop you from testing your code with simple JUnit tests, but are classes that use them still POJOs?

Moreover, other annotations such as EJB 3's `@EJB` annotation, which injects an EJB, let you write code that's difficult to test without running it in the EJB container.

```
class SomeServiceImpl {
    @EJB private AccountDAO accountDAO;...
}
```

The `@EJB` annotation tells the EJB container to “magically” initialize the private field on creation. Not a very POJO-like operation. Similarly, other frameworks' annotations couple your code to the database schema.

These kinds of annotations are definitely handy but using them will couple your application to the framework. That makes it difficult to support multiple frameworks simultaneously. Migrating to a later version or to a new framework can be difficult because you might have to change all of the annotations. This isn't quite as bad as rewriting the code but you still have to change the Java source and most importantly, you have to recompile. If you want to decouple your application from evolving infrastructure frameworks then you might want to consider using pure POJOs and XML metadata.

## Summary

Developing with POJOs and non-invasive frameworks such as Spring, Hibernate, JDO, and EJB 3 provide many benefits.

POJOs simplify and accelerate development. They make it easier to test classes in isolation without deploying them in an application server, a process that often slows down the edit-compile-debug cycle. And because POJOs are decoupled from the infrastructure frameworks that provide services such as transaction management and persistence you can focus on developing the business logic and address those concerns later.

POJOs also make it easier to upgrade your application to newer version of a framework or switch to a different framework entirely. They give your application increased immunity from the problems caused by rapidly evolving infrastructure frameworks. POJOs enable us to build Enterprise Java applications that last.

## References

- Acegi Security System, <http://acegisecurity.sourceforge.net/>
- Download the sample code from [www.pojosinaction.com](http://www.pojosinaction.com)
- Colyer, A. [http://www.aspectprogrammer.org/blogs/adrian/2004/08/when\\_is\\_a\\_pojo.html](http://www.aspectprogrammer.org/blogs/adrian/2004/08/when_is_a_pojo.html)
- Fowler M. <http://www.martinfowler.com/articles/injection.html>
- Fowler M. <http://www.martinfowler.com/bliki/POJO.html>
- Richardson C. (2005). *POJOs in Action*. Manning
- Richardson C. 2002, <http://www.theserverside.com/articles/article.tss?l=TwoLevelDomainModel>
- Vinge, V. (2000). *A Deepness in the Sky*. Tor Books.

## Acknowledgements

I would like to thank Chris Smith, Michael Yuan, Jennifer Shi, David Vydra, and Robert Benson for commenting on this article. ☺

### Listing 1

```
public interface MoneyTransferService {

    Account transfer(String fromAccountId,
        String toAccountId, double amount);

}

public class MoneyTransferServiceImpl implements
    MoneyTransferService {

    private AccountDAO accountDAO;

    public MoneyTransferServiceImpl(AccountDAO accountDAO) {
        this.accountDAO = accountDAO;
    }

    public Account transfer(
        String fromAccountId, String toAccountId,
        double amount) {
        Account fromAccount =
            accountDAO.findAccount(fromAccountId);
        Account toAccount =
            accountDAO.findAccount(toAccountId);
        fromAccount.debit(amount);
        toAccount.credit(amount);
        return toAccount;
    }

}

public class Account {

    private double balance;
```

```
private String accountId;

    public Account(String accountId,
        double initialBalance) {
        this.accountId = accountId;
        this.balance = initialBalance;
    }

    public double getBalance() { return balance; }

    public String getAccountId() { return accountId; }

    public void debit(double amount) { balance -= amount; }

    public void credit(double amount) { balance += amount; }

}

public interface AccountDAO {

    Account findAccount(String accountId);

    Account createAccount(String accountId,
        double initialBalance);

}
```

### Listing 2

```
@Stateless
public class MoneyTransferServiceImpl
    implements MoneyTransferService {

    @EJB
    private AccountDAO accountDAO;

    ...
}
```



# J2EE Lite with Spring Framework

*For a better application*

by Pankaj Tandon

J2EE applications of late have become weight conscious. The combined burden of EJBs and coarse-grained component design has given the term *test driven design* a new meaning: *technology driven design*! Fortunately a host of lightweight solutions are emerging such as PicoContainer, Spring Framework, Hivemind, Hibernate, Castor, and Webwork. In this article I'll discuss my experience with the Spring Framework and how it can be used to make a J2EE application more maintainable, testable, and better performing.

Although there are many areas that Spring 1.2.4 covers, I will be covering only Spring IoC (Inversion of Control) and Spring AOP (Aspect-Oriented Programming). These two along with the (non-Spring) concept of coding to interfaces form the cornerstone on which other Spring technologies are based.

Using Spring AOP and IoC we will see how you can replace container-provided infrastructure services for the following services, a la carte:

- Logging
- Transactions
- Profiling
- Auditing
- Exception monitoring

We will also see:

- Switching XA data sources for nonXA ones for certain methods
- Accessing stateless session beans (SLSB) via a Spring application context
- Accessing Spring application Contexts from a webapp
- Integration tests using IoC and JUnit
- Unit testing using EasyMock and JUnit

To demonstrate these services we will be using a simple ordering application, *made2order*.

## made2order Application

The *made2order* application showcases the above infrastructure services using the Spring Framework. Figure 1 shows the architecture of the application components.

The application consists of an OrderService interface implemented by the OrderServiceImpl concrete class. The OrderDAO interface is implemented by the JdbcOrderDAO concrete class. Finally the domain objects are represented by the Order and the LineItem classes. These six classes represent the POJO-based core business application (in magenta); the other classes are *Spring* classes (used for infrastructure functionality). JSP are used for presentation, packaged in the Web app, and there are two classes for testing

## Database Design and Business Interface

*made2order* has a very simple database design shown in Figure 2.

The business interface for the *made2order* application is shown

in Listing 1. The OrderServiceImpl class implements the above business methods. Note that the getDao() and setDao(...) methods are needed for "injecting" a dependent DAO (Data Access Object Pattern) into the service (we will see more about this later) <http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html>.

The implementation of these methods deals only with the business logic (and DAO interaction). It *does not* have knowledge of infrastructure services such as:

- Transactions
- Logging of enter/exiting from methods
- Auditing
- Profiling long-running methods
- Reporting when exceptions occur
- Data source switching

We will see how the above services will be applied to the *made2order* application in a non-invasive manner.

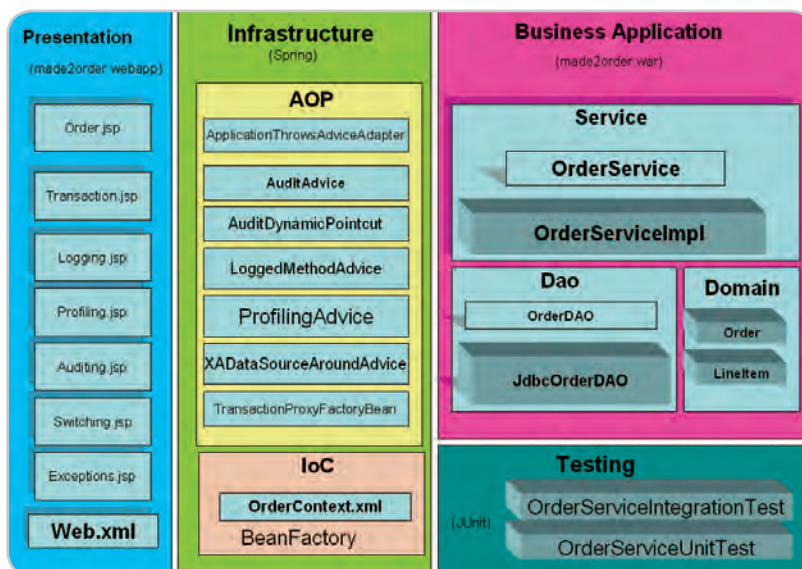


Figure 1 Architecture of the Made2order application

Pankaj Tandon is a software Engineer at Crownncastle. His interests include object-oriented design and development and architecting the software process using open source technologies. He is a Sun Certified Java developer for the Java 2 platform. He lives in Pittsburgh, PA, plays guitar, and mixes music in his spare time.

[pankaj.tandon@gmail.com](mailto:pankaj.tandon@gmail.com)

## Using made2order

The accompanying application is meant to serve as an example for the Spring technologies discussed in this article. The downloadable bundle at [www.jdj.sys-con.com](http://www.jdj.sys-con.com) is a zip file that can be imported into the Eclipse IDE as an Eclipse project.

You can do either one of the following actions (or both) as you read through this article:

- Go through the source code using your favorite IDE and run JUnit tests to understand the examples. The JUnit test suite file (`com.order.acme.OrderServiceIntegrationTest.java`) in the test folder contains several methods to test the above services.
- Install the Web app in a J2EE application server (does not need EJB support) and navigate through the Web pages with the examples.

We will start with the Spring BeanFactory.

## Spring Bean Factory and IoC

The Spring BeanFactory represents a class that manages the life cycle of singleton services (among other things). These services are injected with dependencies

using Inversion of Control (<http://www.martinfowler.com/articles/injection.html>). Services that depend on external resources are “told” what those dependencies are instead of “getting” them using the Service Locator pattern (<http://java.sun.com/blueprints/corej2eepatterns/Patterns/ServiceLocator.html>).

The injection of dependents is achieved either via a constructor or setter injection. The JavaBean that represents the service has a “set” method that accepts the dependant (*setter injection*) or it has a constructor that accepts the dependent object (*constructor injection*). Using an XML configuration file, the service object is configured with its dependent objects using either of these two mechanisms. (The XML file is just one of many formats the configuration can be specified in, others being properties files or just plain Java code.)

There can be one or more bean factories in an application. Bean factories can be overlaid, such that there is one for production but it's overlaid by the one for development.

The bean factory is accessed by the presentation layer to access services offered by the application. In made2order, the OrderService service is needed as a single-

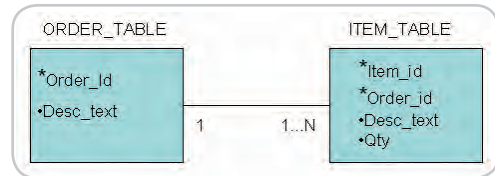


Figure 2 Made2order's database design

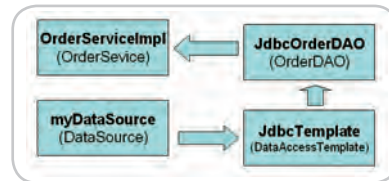


Figure 3 Inversion of Control

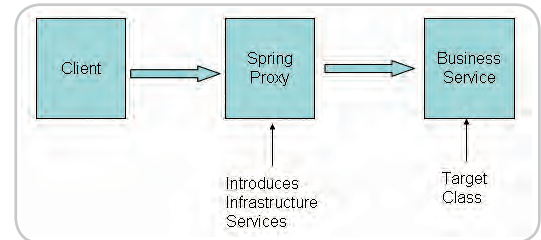


Figure 4 Spring Proxies

ton. The OrderService implementation is dependent on an implementation of OrderDAO. The OrderDAO implementation in turn is dependent on JdbcTemplate ob-

# Bridge the Gap Between Windows and Java

Give your users the freedom to use Microsoft Windows based reporting, analysis, and development applications against your Java data source

OpenAccess™ SDK lets you quickly implement a custom ODBC driver with full SQL support for any data source or application.



## Key OpenAccess™ SDK Features & Benefits

- Write the ODBC driver with SQL support in just weeks - 99% of the required code is contained in the SDK.
- Use Java to implement the data source specific code.
- Guaranteed to work with ODBC compliant applications - hundreds of organizations have used OpenAccess™ to set their data free.
- Fully Unicode enabled for globalization.
- When required easily add JDBC, OLE DB, or .NET drivers.
- Designed from ground up for reliability, scalability, and portability.

To learn more about OpenAccess™ SDK for Java or to try it out, go to [www.OpenAccessSoftware.com/JDJ](http://www.OpenAccessSoftware.com/JDJ) or call 1.888.805.ODBC





ject, (a Spring-supplied helper class that helps with JDBC operations). The JdbcTemplate requires an implementation of a data source. In this case, myDataSource is a data source implementation that is specified in the BeanFactory and injected into the JdbcTemplate.

The BeanFactory returns a singleton of OrderService after “injecting” required dependencies in it as shown in the Figure 3.

Note that an *implementation* is injected into an *interface* at each level.

Figure 3 is represented in the XML file `orderContext.xml` shown in Listing 2. In this listing we see that the BeanFactory defines two data sources. During configuration, either of the data sources is injected into the JdbcTemplate instance.

## Spring Proxies

In the general sense of the word, a proxy is a class that sits between the business class and the client. In the context of Spring, however, a proxy is either a JSE dynamic proxy or a static proxy that implements infrastructure services and is “injected” with a dependency that is the target business class. Then, instead of the client being given a reference to the target service, it’s given a reference to the proxy. In this manner the behavior implemented in the proxy is interjected into the service seen in Figure 4.

With the current release, Spring proxies allow interception at the method level only, whereas some full-blown AOP environments allow field-level interception also. Proxies can be dynamic (JSE dynamic proxies), in which case Java interfaces are proxied; or they can be static, in which case *bytecode* modification occurs. Understandably, static proxies outperform dynamic ones but only marginally.

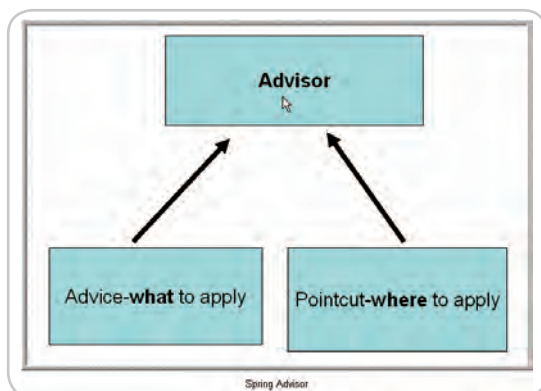


Figure 5 Spring AOP

*Interception* is implemented using concepts from AOP: pointcuts and advice.

## Pointcuts and Advice

**Advice** classes specify the *actual* work to do upon interception whereas **pointcuts** tell Spring *where* to apply advice.

The combination of a *pointcut* and an *advice* is bundled in what Spring calls an **Advisor** (see Figure 5).

We will see Advice, Advisors, and Pointcut classes in action in a made2order application when we apply infrastructure services to the business classes.

The following sections explain how each infrastructure service has been implemented.

### Transaction Service

The OrderService’s `placeOrder(...)`, `modifyOrder(...)`, and `dropOrder()` methods need to be transaction-bound. For this we modify the `orderContext.xml` file by configuring a Spring-supplied transactional proxy called `TransactionProxyFactoryBean`. As shown in the Listing 3, an instance of this proxy is called `myProxiedServiceWithSeveralInterceptors`.

The `TransactionAttributes` section is supplied with regular expressions that represent the methods that need to be transaction bound (`PROPAGATION_REQUIRED`) and those that are not (`PROPAGATION_SUPPORTS`).

Also, the target of this proxy is specified as `orderService`, which is the instance of the `OrderServiceImpl`.

Note that a `transactionManager` is injected into the proxy. An instance of `transactionManager` is set up elsewhere in the configuration file.

The `preInterceptors` are a list of **Advices** and/or **Advisors** that are also tacked onto this proxy in what is called a chain of interceptors. We will see each of these in turn later. For now, note that `preInterceptors` are invoked *before* the target is invoked in the order specified.

### Seeing Transactions in Action Using JUnit

For testing transactions there are two test methods in the test suite: `testNonProxiedServiceForModifyingAnOrder()` and `testProxiedServiceForModifyingAnOrder()`. Both these methods call the `modifyOrder(...)` method on the non-

proxied service in one case and proxied service in the other. The `modifyOrder(...)` method updates the `ORDER_TABLE` table first and then each `lineItem` in the `ITEM_TABLE`. The test method updates an order with two line items. The second `lineItem` has a description whose length is greater than the column width in the database. This forces an error upon update of the line item description in the database. Both the test methods update the `ORDER_TABLE` first and then the first `lineItem` in the `LINE_ITEM` table before throwing an exception while trying to update the long description in the second line item. The proxied method issues a rollback whereas the non-proxied method commits the update to the `ORDER_TABLE` and the first `lineItem` in the `ITEM_TABLE`.

We have thus seen how declarative transactions have been introduced without changing application code.

## Logging Service

It’s sometimes desirable to log entry and exit points in business methods. We will create an advice that will do the logging and then define a pointcut specifying where the advice will need to be applied.

(Note that providing a logging service using Spring is appropriate for entry/exit points from a method, not so much for application-level logs.)

Listing 4 provides the code for `LoggedMethodAdvice`. As we can see, the `LoggedMethodAdvice` implements the AOP Alliance interface (<http://aopalliance.sourceforge.net/>) `MethodInterceptor`, which only has a single method, `invoke(...)`.

The invocation.`proceed()` call is surrounded by the “advice” and therefore this advice is called the “around” advice. The actual logging is achieved via the standard `log4j` calls.

The pointcut is actually a Spring-supplied pointcut called `NameMatchMethodPointcut`. This pointcut accepts (via IoC) a collection of method names to which the advice needs to be applied. This is shown in listing 5.

The advice (`loggedMethodAdvice`) and pointcut (`loggedMethodPointcut`) is combined together as the `LoggedMethodAdvisor` in the `orderContext.xml` file as shown in Listing 6.

Finally, in the BeanFactory (`orderContext.xml` file), the `loggedMethod`

dAdvisor is attached to the existing transactionProxy (myProxiedService-WitheSeveralInterceptors) as a pre-Interceptor.

#### Seeing Logging in Action Using JUnit

The two test methods testLoggingUsingProxiedService() and testLoggingUsingNonProxiedService() can be used to test the Logging service.

Looking at the accompanying log4j file (made2order.log), we see that the proxied method logs statements for the placeOrder(), modifyOrder(), and dropOrder() but not for the showOrder(). The non-proxied method does not log any methods at all.

#### Auditing Service

This service shows how we can use the proxy mechanism to selectively audit methods that are executed by an external user (in our example). Here too we will be implementing AuditAdvice as a MethodBeforeAdvice. However, for the pointcut, instead of using the out-of-the-box NameMatchMethodPointcut (as in the case of logging), this time we will use an instance of DynamicMethodMatcher-

Pointcut. This pointcut is implemented in the class AuditDynamicPointcut (see Listing 7). It allows us to dynamically match class names, method names, and arguments, in that order. In Listing 7 we see that the AuditDynamicPointcut class accepts a collection of methodNames. These are injected into the pointcut in the beanFactory. First the AuditDynamicPointcut examines to see if the class being proxied is assignable from OrderService. If that is true, the next check is to see if the currently executed method's name is in the list of methodNames injected into this proxy. Finally the matches(Method method, Class clazz, Object[] o) method examines the arguments passed to the method and only returns true if the argument passed is false (representing externalUser = true).

We also see that in the orderContext.xml file (see Listing 8), the property methodNames is set to printReport signifying that the pointcut should return a true only if the methodName is printReport as shown in the implementation in Listing 8. The advice is tied to the pointcut resulting in an Advisor as shown in Listing 9. (Listings 9-20 and additional source code

can be downloaded from <http://jdj.sys-con.com>.)

As a result, this pointcut will apply the auditAdvice to the printReport method of OrderService *only* if the argument passed to printReport is false. In any other case, the advice will not be applied.

Although in made2order auditing is implemented as log4j output, note that AuditAdvice can be also be implemented by writing to the database, as a *part of a separate transaction*.

#### Seeing Auditing in Action Using JUnit

The test suite OrderServiceIntegrationTest's two methods testAuditingUsingProxiedService() and testAuditingUsingNonProxiedService() are executed. Both the methods make calls to the business method printReport(...), passing in an argument of internalUser the first time and an externalUser the second time. We see that:

- The auditing statement is only logged on the proxied service.
- The auditing statement is only logged on the proxied service when an externalUser makes that call.



  
DynamicPDF  
[www.dynamicpdf.com](http://www.dynamicpdf.com)

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

#### DynamicPDF™ Merger v3.0

Our flexible and highly efficient class library for manipulating and adding new content to existing PDFs is available natively for Java

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

#### DynamicPDF™ Generator v3.0

Our popular and highly efficient class library for real time PDF creation is available natively for Java

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

Try our free Community Edition!

 ceTesoftware  
INFINITE POSSIBILITIES  
800.631.5006 +1 301.656.1331

## Profiling Service

The profiling of methods is achieved via the ProfilingAdvice class (see Listing 10). Here, for illustration only, the use of a pointcut is avoided, so that *every* method that has the word “Long” in its name is intercepted. This is *not* the preferred way of interception as it’s slower performing than interception using a pointcut.

Since there is no pointcut associated with this advice, it is directly added to the list of interceptors (without a corresponding advisor) to the TransactionalProxy (see Listing 11). The behavior of this service is similar to the Logging and Auditing service. Further explanation is omitted for the sake of brevity.

## Reporting of Exceptions

It’s often desirable to track certain exceptions in a production environment via e-mail notification or logging. We will see here how exceptions can be handled declaratively using Spring proxies.

The advice that is needed for this functionality implements the Spring-provided ThrowsAdvice interface. This tag interface requires the implementation of only one method called afterThrowing(...). The last argument to this method is of type Throwable. Hence, different behavior can be implemented for different exceptions as shown in Listing 12.

This advice, like the Profiling Advice, does not have a pointcut associated with it. However, unlike the Profiling Advice where the logic to test the method name existed within the advice, the ThrowsAdvice is applied to *any* method throwing the mentioned exceptions.

## Seeing Reporting of Exceptions in Action Using JUnit

From the test suite, run testProxiedServiceForExceptionReporting(). This method calls the printLongReport(...) method on the proxied orderService with three different arguments. The first two arguments cause the NullPointerException and the IllegalStateException, whereas the last call throws a generic RuntimeException. We see by looking at the log4j output that the ThrowsAdvice is applied in the first two cases but not in the third.

## Data Source Switching

Enterprise applications may sometimes write to different databases. In this case the transaction that represents the unit-of-work will require an XA driver that can handle transactions across the different databases. This driver is understandably not as fast-performing as its plainer counterpart which deals with transactions spanning only one type of database.

Made2order illustrates a good solution to this problem by combining the power of IoC and AOP. Using IoC, the orderService is injected with a plain, non-xa data source, but, using AOP, the data source is switched to an xa one for certain methods that are configured declaratively.

First, in the orderContext.xml file, both the xa and non-xa data sources are injected into the XADataSourceAroundAdvice. (Note that the two dataSources are declared elsewhere in the BeanFactory.)

Then, the XADataSourceAroundAdvice is implemented as shown in Listing 14. Note how the dao instance is gotten from the service, then its data source

is switched before the invocation and switched back after the invocation.

Next, the xaMethodPointCut is defined as a NameMatchMethodPointcut and two methods, placeOrder(...) and modifyOrder(...), are declared to be targets for the data source switch (see Listing 15). Finally, the advice and pointcut are combined to produce an Advisor as in Listing 16.

## Seeing Data Source Switching in Action Using JUnit

In the test suite OrderServiceIntegrationTest, we can execute the method testDataSourceSwapping(). This method invokes the placeOrder(...), modifyOrder(...) and then the dropOrder(...). Looking at the log output, we see that the data source is swapped for the first two methods but not for the last method.

## AOP and IoC in Concert

AOP and IoC are a synergetic combination of concepts. Where the BeanFactory provides us with a single point of reference for our service objects and other singleton resources, AOP allows us to elegantly decorate those references with interceptors.

The example of this in the made2order above can be enhanced thus: if at a later time another service’s method (PurchasingService.cancelPurchase()) is modified to invoke OrderService.dropOrder(), and each is involved in writing to different databases, then PurchasingService.cancelPurchase() is a good candidate for xa resource swapping, while leaving OrderService.dropOrder() to use the default non-xa resource. That resource swapping can be achieved with the same ease of configuration and elegance as was done with OrderService.placeOrder() by making PurchasingService a Spring managed Bean with advised proxies.

At this point we have applied the infrastructure services of transactions, logging, auditing, profiling, exception monitoring, and data source swapping to our business interface and, at the same time, have not convoluted our business code with references to these services. Also we have shown how we can easily change the infrastructure services to apply (or *not* apply) to a set of methods via configuration changes to the pointcut parameters in the BeanFactory.

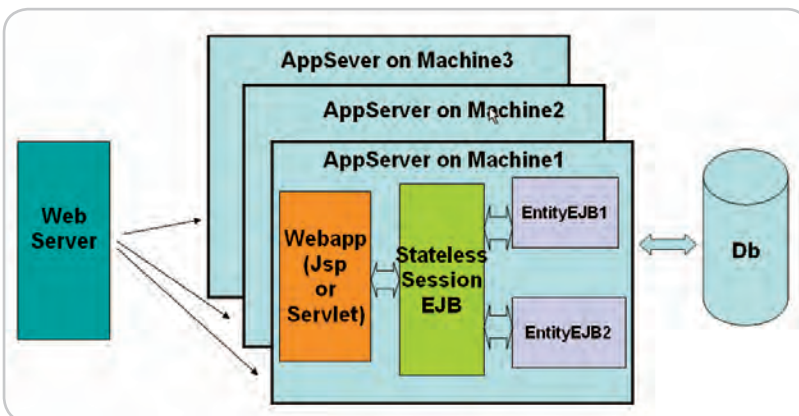


Figure 6 Physical architecture of an EJB J2EE application



## Spring and EJBs

We can now look at EJBs and discuss how EJBs play with Spring technologies. Since providing infrastructure services is the value proposition for both technologies (at least, in part), it would seem that there is an overlap. Let us look a little closer at the runtime services that the EJB container provides:

- Remoting
- Clustering
- Security
- Caching (in case of entity EJBs)
- Persistence (in case of entity EJBs)
- Transactions

Application servers typically bundle all the above infrastructure services in one EJB container. Our applications therefore end up being affected in two major ways:

1. Applications have to carry the extra baggage of loading up the entire container even if it has to use only some of the services offered.
2. Business objects are more coarsely designed to accommodate bandwidth limitations that come with EJBs being accessed remotely. (Typically you will not have an EJB for Order and another for LineItem.) Because of the invasive nature of EJBs (business classes have to extend from EJBObject/EJBRemote), you lose out on the possibility of a fine-grained OO design.

To further drive this point home, let's assume a physical architecture of a J2EE application that uses the EJB approach. (This by no means is the only possible architecture, but it helps make some important points.) Figure 6 shows Web apps and some EJBs collocated on the same physical machine.

Just because Web apps and EJBs are collocated, three of these runtime services are not needed:

- The **remoting** of EJBs is not necessary as Web apps will call EJBs locally.
- **Clustering** of EJBs is achieved via replica-aware home and remote stubs. Again, since the caller (Web app) and the called object (EJB) are on the same machine, there is not much sense in using cluster-aware stubs. Note that HttpSession replication for making Web apps cluster-aware is still desirable though.
- Typically, if EJBs are not remototed, then you wouldn't want to impose **security** restrictions at the EJB tier.

As for the following runtime services:

- **Caching** of entity EJBs in the container, on the other hand, is a valuable benefit that may be considered depending on the use case at hand.
- **Transactions** are the most commonly used service provided declaratively by the EJB container. Many developers prefer the declarative use of Container Managed Transactions (CMT) over programmatic transaction management.
- **Persistence** is now a separate specification from the EJB specification (JSR-220) and therefore, in future, will not be a direct service of the EJB container. However, it is a valuable service that can save development time, and, in some cases perform better.

We see that just collocating causes some of the services provided by the EJB container to become redundant.

Therefore applications designed using EJBs will have to interject a container every time a service is invoked, even if it doesn't need 90% of the services it offers. In addition, it will be designed coarsely with the EJB's remote interface and bandwidth considerations. Compare this with a Spring-based design where the business classes are first designed, unencumbered by EJB technology, using all the power of OO design like inheritance and coding to interfaces. Once the business design is complete, *then* infrastructure services are applied to them on an as-needed basis.

However, many times collocation is *not* a good fit. Sometimes, depending on the use case, EJBs become a desired feature. For example, middle-tier bean caching necessitates the use of entity EJBs and clustering at an EJB level necessitates the use of session EJBs. For these and other reasons, let's look at how to access SLSBs from Spring.

## Defining StatelessSessionBeans in the BeanFactory

SLSBs are widely used to provide the service façade in business applications. Sometimes it's not advisable to remove all SLSBs in favor of proxied objects. In fact, the non-invasive nature of Spring makes it a good candidate for phasing in its use. For that reason or other legitimate uses of SLSBs (like third-party SLSBs), it's easy to make Spring managed beans and SLSBs coexist in the same application.

## Build interactive diagrams easier than you ever imagined

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



**FREE Download  
With Full Support!**

**Go** **Diagram**

Interactive diagram components

[www.nwoods.com](http://www.nwoods.com)

In Listing 17 we will see how an SLSB can be wrapped in a Spring-managed bean that can be retrieved from the BeanFactory. In the listing the `orderContext.xml` file shows how a Spring-supplied class (`SimpleRemoteStatelessSessionProxyFactoryBean`) is injected with a `jndiName` and a JNDI location for telling it where a certain EJB lives.

The second point is the `businessInterface` property into which `OrderService` interface is injected. This is different from the `EJBRemote` interface and is just a Java interface that is also implemented by the POJO manager `OrderServiceImpl`.

Beyond that, it is business as usual. The SLSB can now be accessed just like any other bean managed by the Spring bean factory by using the `handleOrderServiceEJBProxy`.

At this point, we have demonstrated that Spring can easily straddle an application that uses SLSBs at the service layer and service objects built from scratch using Spring.

### Accessing the Bean Factory from a Web App

Now that we have defined services in the bean factory, how do we actually access the instance of those services? Spring provides several generic classes to make this task easier.

The `ApplicationContext` interface is at the root of the hierarchy of several interfaces whose implementers read the `BeanFactory` (or factories) into objects accessible by clients. The `WebApplicationContext` is one such interface for accessing the `BeanFactory` from a Web app, as shown in Listing 18.

First the `web.xml` of the Web app has the following lines supplied that tell the `ServletContext` about Spring's bean factory. The `ContextLoaderListener` loads up the `BeanFactory` configuration at Web app startup time. Next, the `Order.jsp` (that lives in the bundled Web app `made2order.war`) loads up the context from the `ServletContext` as shown in Listing 19. `orderService` and `myProxiedServiceWithSeveralInterceptors` are instances of singleton services that are defined in the `BeanFactory`.

Similarly, `ClassPathXmlApplicationContext(paths)` is another helper class that gets services from the `BeanFactory` as is used in the JUnit test suite's `setup()` method (see Listing 20). Note that a new `BeanFactory` can be swapped in at runtime by placing it in the classpath and destroying the current context.

### Testing

Encouraging good testing practices is one of the value propositions of Spring. IoC is a good mechanism to encourage integration testing, whereas coding to interfaces is really what facilitates Unit testing.

In our example, `jdbcOrderDAO` can be easily replaced by a `hibernateOrderDAO` for conducting JUnit integration tests from within Eclipse, for example.

Similarly, instead of using a POJO manager implementation of `OrderService`, you can just as easily replace the implementation with the `OrderEJB` (which also implements the `OrderService` interface).

For unit testing on the other hand, **EasyMock** has been used. The class `OrderServiceUnitTest` *only* tests the service implementation by injecting a mock dao into the service layer. Since `OrderDAO` is an interface, this is easily achieved by casting the `MockControl.getMock()` method to return an `OrderDAO` instance. Then, by injecting the mockDAO to the service (setter injection), we can only test the code in the service. **EasyMock** uses the concept of recording and playback. The exact calls to the mockDAO are recorded first and then the test case is run. **EasyMock** asserts whether or not the playback is exactly the same as the one recorded.

For more detailed examples, look at the class `OrderServiceUnitTest.java`.

Because of the simple nature of this application, it certainly doesn't make it a good candidate for an example of test driven design. However, all else being equal, easily testable code will more be more likely to drive design than code that is hard to test.

### Where Do We Go from Here

Spring provides an IoC and AOP-based framework to build infrastructure services. Even though we identified some EJB container-based services that are not needed; sometimes, depending on the use case, those services provide value. In those cases, remoting, security, and persistence can be applied to Spring proxy using the same mechanism that was used by transactions or auditing. This allows for open-ended growth where services are applied on an as-needed basis to certain Spring-managed service beans. The sample application does not show these additional features, however,

the interested reader can explore Spring supplied support classes for:

- Remoting using Caucho's Hessian and Burlap protocols
- Security using Acegi
- Persistence using Hibernate

There are several Spring classes for other services, which are not mentioned here.

### Summary

We have implemented the separation-of-concerns pattern by isolating business code from infrastructure code via method interception and Spring proxies. In addition, using IoC and the convenience of a `BeanFactory`, we are able to inject different implementations into different service contracts.

By moving to a POJO-based business tier, our design is not encumbered by bandwidth considerations as with EJB design. The design of the application can be as object-oriented and fine-grained as needed because we are not dealing with issues of multiple inheritance or layering.

We looked at the scalability issues and concluded that we can apply several enterprise-level services on an as-needed basis while still remaining cluster-able at a Web app level.

We looked at testing strategies and showed that coding to interfaces allows for the easy use of third-party testing tools like **EasyMock**. IoC via interfaces allows for integration testing using a configurable bean factory.

In the end, `made2order` is **easier to maintain** because we have removed infrastructure code from our business code. It is **easier to test** because of ease of configuration using a bean factory approach and coding to interfaces. Last, it **performs better** compared to an EJB-based solution because we are not interjecting a one-size-fits-all container in our business operations. ☺

### References

- [www.springframework.org](http://www.springframework.org)
- [www.easymock.org](http://www.easymock.org)
- [www.juint.org](http://www.juint.org)
- [maven.apache.org](http://maven.apache.org)
- [www.caucho.com/](http://www.caucho.com/)
- [acegisecurity.sourceforge.net/](http://acegisecurity.sourceforge.net/)
- [www.hibernate.org/](http://www.hibernate.org/)
- <http://sourceforge.net/projects/aopalliance>
- [www.martinfowler.com/](http://www.martinfowler.com/)

**Listing 1: OrderService business interface**

```

*/
public interface OrderService {
    public OrderDAO getDao() ;
    public void setDao(OrderDAO dao) ;
    public ArrayList showAllOrders();
    public int countAllOrders();
    public Order showOrder(int orderId) ;
    public void placeOrder(Order order) ;
    public void modifyOrder(Order order);
    public void dropOrder(int orderId);
    public void printReport(boolean internalUser) ;
    public void printLongReport(int longRunning);
}

```

**Listing 2: orderContext.xml file representing the BeanFactory**

```

<bean id="orderService" class="com.acme.order.manager.
OrderServiceImpl" >
    <property name="dao" ref="orderDAO" />
</bean>

<bean id="orderDAO" class="com.acme.order.dao.jdbc.JdbcOrderDAO">
    <property name="jdbcTemplate" ref="orderJdbcTemplate" />
</bean>

<bean id="orderJdbcTemplate" class="org.springframework.jdbc.
core.JdbcTemplate" >
    <property name="dataSource" ref="myDataSource"/>
</bean>

. . .

<!-- non XA dataSource (right now the same) -->
<bean id="xaDataSourceFromWLS" class="org.springframework.jndi.
JndiObjectFactoryBean">
    <property name="jndiName">
        <value>${jdbc.jndi.pool}</value>
    </property>
    <property name="jndiEnvironment">
        <props>
            <prop key="java.naming.factory.initial">${java.naming.factory.
initial}</prop>
            <prop key="java.naming.provider.url">${java.naming.provider.
url}</prop>
        </props>
    </property>
</bean>

```

**Listing 3: Transaction ProxyBean Configuration**

```

<bean id="myProxiedServiceWithSeveralInterceptors"
class="org.springframework.transaction.interceptor.Transaction
ProxyFactoryBean">
    <property name="target" ref="orderService"/>
    <property name="transactionManager" ref="transactionManager"/>
</bean>

<property name="transactionAttributes">
    <props>
        <prop key="place*">PROPAGATION_REQUIRED</prop>
        <prop key="modify*">PROPAGATION_REQUIRED</prop>
        <prop key="drop*">PROPAGATION_REQUIRED</prop>

        <prop key="show*">PROPAGATION_SUPPORTS</prop>
        <prop key="count*">PROPAGATION_SUPPORTS</prop>
        <prop key="print*">PROPAGATION_SUPPORTS</prop>
        <prop key="*">PROPAGATION_REQUIRED,readOnly</prop>
    </props>
</property>
<!-- force use of CGLIB for performance -->
<property name="optimize">
    <value>true</value>
</property>
<property name="proxyTargetClass">
    <value>true</value>
</property>
<property name="frozen">
    <value>true</value>
</property>

<property name="preInterceptors" >
    <list>
        <ref local="xaConnectionAdvisor"/>
        <ref local="auditAdvisor"/>
        <ref local="loggedMethodAdvisor"/>
        <ref local="profilingAdvice"/>
        <ref local="throwsAdvice"/>
    </list>
</property>
</bean>

```

**Listing 4: LoggedMethodAdvice**

```

public class LoggedMethodAdvice implements MethodInterceptor{
    protected final Logger LOG = Logger.getLogger(getClass().get-
Name());
    public Object invoke(MethodInvocation invocation) throws
Throwable{
        Method method = invocation.getMethod();
        String className = invocation.getThis().getClass().getName();
        String methodName = method.getName();
        LOG.info("Going to invoke: " + className + ":" + methodName);
        Object ret = invocation.proceed();
        LOG.info("Done with method: " + className + ":" + methodName);
        return ret;
    }
}

```

**Listing 5: LoggedMethodPointcut configuration**

```

<bean id="loggedMethodPointcut" class="org.springframework.aop.
support.NameMatchMethodPointcut" >
    <property name="mappedNames">
        <list>
            <value>placeOrder</value>
            <value>modifyOrder</value>
            <value>dropOrder</value>
        </list>
    </property>
</bean>

```

**Listing 6: LoggedMethodAdvisor configuration**

```

<bean id="loggedMethodAdvisor" class="org.springframework.aop.sup-
port.DefaultPointcutAdvisor" >
    <constructor-arg index="0">
        <ref local="loggedMethodPointcut"/>
    </constructor-arg>
    <constructor-arg index="1">
        <ref local="loggedMethodAdvice"/>
    </constructor-arg>
</bean>

```

**Listing 7: AuditDynamicPointcut**

```

public class AuditDynamicPointcut extends DynamicMethodMatcherPo-
intcut {

    ArrayList methodNames;

    public boolean matches(Method method, Class clazz, Object[]
object) {
        boolean ret = false;
        if ((object != null) && (object.length > 0)){
            boolean isInternal = ((Boolean)object[0]).booleanValue();
            //Only audit those calls that are not from internal users
            ret = !isInternal;
        }
        return ret;
    }

    public boolean matches(Method method, Class clazz) {
        String methodName = method.getName();
        boolean ret = false;
        if (methodNames.contains(methodName)){
            ret = true;
        }
        return ret;
    }

    public ClassFilter getClassFilter(){
        return new ClassFilter() {
            public boolean matches (Class clazz){
                return (OrderService.class.isAssignableFrom(clazz));
            }
        };
    }

    public ArrayList getMethodNames() {
        return methodNames;
    }
}

```

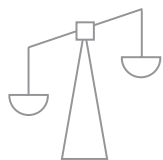
**Listing 8: Specifying auditable methods in the BeanFactory declaratively**

```

<bean id="auditPointcut" class="com.acme.advice.
AuditDynamicPointcut" >
    <property name="methodNames">
        <list>
            <value>printReport</value>
        </list>
    </property>
</bean>

```





# Spring and EJB 3.0 in Harmony

by Aleš Justin

*In search of the best of both worlds*

The new EJB 3.0 specification supports some notion of dependency injection via annotations. As an avid Spring user, I'm used to configuring fine-grained beans with Spring bean factories and XML. How does EJB 3.0 compare? More importantly, can we use EJB 3.0 POJOs and Spring POJOs side-by-side in applications? In this article, I'll try to answer those questions based on my own investigations and experiences. As it turns out, using a versatile application server like the JBoss Application Server (AS), Spring and EJB 3.0 POJOs can co-exist in harmony in your applications.

## EJB3 Dependency Injection

In EJB 3.0 it's fairly easy to inject Java EE components into an EJB object using various annotations provided by the specification. `@Resource` is available to inject things like datasources and JMS destinations. `@EJB` is there to get references to other EJBs. `@PersistenceContext` can get you a reference to the EntityManager service. You can use these annotations to inject directly into the field of your EJB, or you can apply them on setter methods. Here are some examples:

```
@Stateful
public class ShoppingCartBean implements
ShoppingCart {
    @Resource(name="DefaultDS") private
DataSource ds;
    @EJB private CreditCardProcessor ccp;

    EntityManager em;

    @PersistenceContext(unitName="orderdb")
    public void setEntityManager(EntityManager
em) {
        this.em = em;
    }
}
```

For developers who don't want to use annotations to inject components, EJB 3.0 provides an XML alternative to injection annotations:

```
<ejb-ref>
  <ejb-ref-name>ejb/CreditCardProcessor</
ejb-ref-name>
  <ejb-ref-type>Session</ejb-ref/type>
  <local>com.acme.CreditCardProcessor</
local>
  <ejb-link>CreditCardEJB</ejb-link>
  <inject-target>ccp</inject-target>
</ejb-ref>
```

Great! So we have two ways of injecting dependencies in EJB 3.0. However, one big problem with EJB 3.0 dependency injection is that there's no way of configuring, defining, and injecting plain Java beans. You can only inject Java EE component types and `<env-entry>` only lets you inject primitive types. Sure, EJBs look a lot like POJOs and JBoss has the concept of a `@Service` bean, which is a singleton, but do we really have to define and inject these component objects that have to run in containers? And what about beans from third-party libraries and pre-existing code. There must be something better! Could it be JNDI registration? Maybe, but who will instantiate and construct our beans? Really this is what Spring's dependency injection is all about. But how do we get Spring's beans into an EJB3 container?

## Spring and EJB3 – Synergy or Superfluous?

To integrate Spring beans with EJBs, the application server plays the central role since it can deploy both components. JBoss AS seems to be the best platform here because it was the first application server to support EJB 3.0

and it is an Open Source product. More importantly, JBoss AS's microkernel architecture is flexible enough to create any package/component type that can be deployed in the JBoss runtime. JBoss AOP and interceptor technology could be used to glue custom behavior into POJOs and/or EJBs.

So I decided to write a Spring deployer for JBoss AS. In the rest of the article, I'll discuss exactly what I did and how you can use the deployer. As it turns out, there really is synergy between Spring and JBoss EJB 3.0. Developers can have the best of both worlds.

## JBoss AS Deployers

The first challenge is to package Spring into a deployable unit to eliminate the bootstrapping code that's required in most Spring applications. The deployer should work with a custom Spring archive type in much the same way Java EE has .EARs and .WARs. You can define new archive types in JBoss by writing a JBoss Deployer.

When JBoss boots up, it scans the deploy directory for available packaged components that are JBoss services or your packaged applications. JBoss Deployers are responsible for managing these packaged components. Their purpose is to understand deployment descriptors and archive formats so they can be deployed into the JBoss application server at runtime. Deployers are also responsible for creating classloaders for packaged component and managing hot deployment. There's a specific deployer service for each component type in JBoss: enterprise archives (.ear), Web archives (.war), EJB archives (.jar), datasources (-ds.xml), Hibernate archives (.har), etc. I wanted to be able to define a Spring archive (.spring) that would look like this:

Aleš Justin is a Java developer at Genera Lynx d.o.o. He's also a JBoss contributor and committer.

ales.justin@genera-lynx.com

```
myapp.spring/
  org/acme/MyBean.class
  META-INF/spring.xml
```

It would really look like an EJB jar, but instead of a ejb-jar.xml deployment descriptor, one would be specific to Spring. The JBoss Spring deployer recognizes the myapp.spring deployment, parses the deployment descriptor to create a Spring bean factory, and registers that Spring bean factory under some JNDI name so that it can be looked up and used by other applications. And if this archive was removed, JBoss should destroy the bean factory and unregister it from JNDI. Basically, the goal is to let Spring bean factories be hot-deployable in a JBoss environment.

JBoss has an abstract API for writing new deployers and it was very simple to write a Spring-based archive type. JBoss's deployer API provides a simple abstract class – org.jboss.deployment.SubDeployerSupport – that I used to implement my Spring Deployer. One simply has to implement or override a few methods to get things working. Let's look at those methods:

```
protected void initializeMainDeployer() {
    setSuffixes(new String[]{"spring", "-spring.xml"});
    setRelativeOrder(350);
}

/* @return True if this deployer
 * can deploy the given DeploymentInfo.
 * @jmx:managed-operation
 */
public boolean accepts(DeploymentInfo di) {
    String urlStr = di.url.toString();
    return urlStr.endsWith("spring") ||
        urlStr.endsWith("spring/") ||
        urlStr.endsWith("-spring.xml");
}
```

This code is trivial but it's the core of determining if a deployed component is a Spring component. Each JBoss deployer is asked if it *accepts* a given archive for deployment. This code specifies that the Spring Deployer accepts file names that end with .spring or -spring.xml. JBoss has a default ordering schema when it deploys packages. Since deployers are themselves packaged components, they tell the JBoss runtime what relative order their component type should be deployed in as shown in Listing 1.

As I said before, JBoss deployers support the hot redeployment of your custom

component type at runtime. To enable this, you have to give JBoss a URL to watch to see if the timestamp on this file changes at runtime. Our deployer will support raw Spring XML files, .spring JAR archives, and exploded .spring archives. The init() method above sets up the watch URL as shown in Listing 2.

Deployers create() method calls BeanFactoryLoader's create method, which is responsible for managing our different Spring bean factories. When instantiating beans in a new bean factory we make sure that newly created beans are loaded by a global scope classloader. Normal JNDI bindings require a serializable object, which is unusable for our Spring bean factory. JBoss provides a JNDI ObjectFactory implementation that lets us store objects in a non-serializable form in a local JNDI context. The binding will only be valid as long as our bean factory is deployed.

Since we'd like to hot deploy many different Spring archives, each of them must be uniquely represented in our environment. For that purpose, JNDI will be used as a registry for these deployments. But where can we get our local JNDI name? By default, this JNDI name is obtained from the archive's file name: <name>.spring or <name>-spring.xml. But since it's also possible for each bean factory to have parent bean factory, there should be a way to define this parent's name too. This can be done in Spring's beans xml descriptor in description tag as shown in Listing 3.

We are parsing this description tag and looking for a regular expression pattern of 'BeanFactory=(<name>)' and 'ParentBeanFactory=(<name>)' so see Listing 4.

Every time a Spring component is undeployed, we get the corresponding Bean factory through the URL string key in our bean factory map. We remove the map entry, destroy the bean factory singletons, and unbind it from JNDI. When undeploying components be careful of the bean factory hierarchy; don't undeploy some other component's parent bean factory. When shutting down, JBossAS undeploys components in reverse order, which is the right behavior for our child-parent bean factory hierarchy.

As you could see throughout the code, a lot of methods are described as JMX-managed. You've probably heard what the JBoss AS microkernel is all about. It's a JMX server. So what else would you expect; the deployers are also JMX MBeans. Each MBean, or call it service, must have its descriptor file. In JBoss that file is META-INF/jboss-service.xml:

```
<server>
  <mbean code="org.jboss.spring.deployment.SpringBeanFactoryDeployer" name="jboss.spring:service=SpringBeanFactoryDeployer" />
</server>
```

Or you can use the whole power of Spring's application context (postprocessors, message source, event multicaster, listeners, ...) class by changing the content of our jboss-service.xml file to:

```
<server>
  <mbean code="org.jboss.spring.deployment.SpringApplicationContextDeployer" name="jboss.spring:service=SpringApplicationContextDeployer" />
</server>
```

So now our Spring Deployer is coded up and we have to package it. JBoss provides a .deployer archive type. When you put this archive type in the JBoss deploy directory, JBoss will automatically be able to use the new archive type you've defined in your new deployer. Our Spring deployer archive structure looks like this:

```
jboss-spring-jdk5.deployer/
  jboss-spring-jdk5.jar
  spring-beans.jar
  spring-context.jar
  spring-core.jar
  META-INF/
    jboss-service.xml
```

## Injecting Spring into EJBs

So now we have a way to package Spring and get it into the JBoss environment. It's time to write the "glue" that injects Spring beans into an EJB. The idea would be to define a new annotation to do this:

```
@Target({ElementType.METHOD, ElementType.FIELD})
@Retention(RetentionPolicy.RUNTIME)
public @interface Spring {
    String jndiName();
    String bean();
}
```

The annotation would be used inside an EJB this way:

```
@Stateless
public class MyBean implements MyRemote {

    @Spring(jndiName="myApp", bean="SomeBean")
    private SomeBeanClass pojo;

}
```

So how do we get the JBoss EJB 3.0 container to understand the @Spring annotation and do the injection for us? The answer is JBoss AOP. How to write such an aspect? Let's walk through it step-by-step.

The first thing to do is write an interceptor that has all of our Spring injection logic. Let's look at the implementation in Listing 4.

This interceptor will intercept the construction of the EJB. The inject() method is responsible for getting the class of the object and reflecting on it to find the setter methods and/or fields that have the @Spring annotation attached to it. From the information in the @Spring annotation, the interceptor looks up the bean factory in JNDI and injects the bean into the annotated method or field. I won't go into the boring details of the inject() method, but do note that it's encapsulated in the SpringInjectionSupport base class. Later in this article, I'll show you a more portable way of defining this Spring injection annotation so that it can be used in other EJB 3.0 containers as well as in JBoss.

So now that the interceptor is defined, we need a way to bind it into the EJB environment. To do this, you need to edit the ejb3-interceptors-aop.xml file in the JBoss deploy directory. This file contains all defined interceptor and interceptors bindings used by all the EJB containers. We'll have to add a declaration of our Spring interceptor there:

```
<interceptor
  class="org.jboss.spring.interceptor.
  SpringInjectionInterceptor"/>
```

Next we have to bind this interceptor to each EJB container definition. If you look at the ejb3-interceptors-aop.xml file you'll see a bunch of AOP domains defined for each container type. You'll have to add the following to each domain:

```
<bind pointcut="execution(*->new(..))">
  <interceptor-ref name="org.jboss.
  spring.interceptor.SpringInjectionInterce
  ptor"/>
</bind>
```

The SpringInjectionInterceptor will now be invoked every time an instance of an EJB is created. But what if the @Spring annotation isn't used? Isn't this interceptor additional overhead?

JBoss AOP has ways of defining a more complex pointcut expression to avoid applying the interceptor, but I won't get into the details here. Check out the JBoss AOP documentation for more details.

An interesting thing about this interceptor and XML bindings is that they can be reused outside of EJB 3.0. If you use a full JBoss AOP, you can use the @Spring annotation to inject Spring-defined beans into *any* plain Java Class. You're not limited to using this annotation to EJBs.

That's it! We've integrated JBoss, Spring, and JBoss EJB 3.0.

### Portability with EJB 3.0 Interceptors

One problem with using the JBoss AOP approach to integrating EJB 3.0 and Spring is that you have to use JBoss AOP. Sure JBoss AOP is usable outside of the JBoss application server, but do I really want a full-blown AOP implementation just to enable injection into my EJBs? Luckily, the EJB 3.0 specification provides a way to do this portably in any vendor's EJB 3.0 implementation.

EJB 3.0 has the concept of interceptors. Not only can you intercept methods, but you can also intercept EJB callback events like @PostConstruct (ejbCreate for you EJB 2.1 users). To intercept callbacks, you have to write something called an Interceptor. The Interceptor is a plain Java class with methods annotated with the callback event you're interested in intercepting. These methods take one parameter, the bean instance whose callback you are intercepting. What we can do is write one of these interceptors to do the same kind of injection processing we did with a JBoss AOP interceptor:

```
public class SpringLifecycleInterceptor
  extends SpringPassivationInterceptor {

    @PostConstruct
    public void postConstruct(Object bean)
  throws Throwable {
        inject(bean);
    }
}
```

We can apply this interceptor using an annotation on the bean class:

```
@Stateless
@Interceptors(value = SpringLifecycleInterce
```

```
ptor.class)
public class MyBean ... {
...
}
```

Alternatively, a partial XML deployment descriptor can be used to apply the callback:

```
<interceptor-binding>
  <ejb-name>MyBean</ejb-name>
  <interceptor-class>
    org.jboss.spring.SpringLifecycleIntercepto
  r
  </interceptor-class>
</interceptor-binding>
```

It would be quite annoying to have to apply this interceptor to each and every bean class or define it for each bean in XML, so there in the final draft of the EJB 3.0 specification will be a way to define default interceptors.

### Conclusion

JBoss Deployers lets me create a new Spring archive type that's hot-deployable as-is into the JBoss runtime. That in and of itself is a simple, but powerful way to bootstrap Spring into the JBoss environment without having to write any of the specific bootstrap code that you normally have to write when using Spring.

EJB 3.0 greatly simplifies the development of enterprise applications. Annotations make it very easy to define and deploy EJBs without having to use XML. EJB 3.0 has some dependency injection, but it's not complete. Spring can be used as a compliment to EJB 3.0 to fill in the dependency injection gaps in the Java EE specification. With JBoss AOP or interceptors, I could easily define a new @Spring injection annotation to inject these deployed Spring beans directly into an EJB. ☺

### Resources

The JBoss Spring Deployer and EJB 3.0 integration is a free download: [http://sourceforge.net/project/show-files.php?group\\_id=22866&package\\_id=161914](http://sourceforge.net/project/show-files.php?group_id=22866&package_id=161914)

JBoss has also created a new Spring Integration forum where you can talk about other ways you'd like Spring to be integrated into the JBoss environment: <http://www.jboss.org/index.html?module=bb&op=viewforum&f=223>



# 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](http://www.ajaxseminar.com)

March 13, 2006

**Marriott**

Marriott Marquis Times Square  
New York, NY

April 24, 2006

**DoubleTree**

DoubleTree Hotel  
San Jose, CA

For more information  
Call 201-802-3022 or  
email [events@sys-con.com](mailto:events@sys-con.com)

# REAL WORLD

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.

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

*Jeremy Geelan*

Jeremy Geelan  
Conference Chair, Real-World AJAX  
[jeremy@sys-con.com](mailto:jeremy@sys-con.com)



**Jesse James Garrett**  
Creator of WebLogic,  
Ph.D., President and  
CTO, Zimbra



**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)	

MEDIA SPONSOR



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

PRODUCED BY  
SYS-CON  
EVENTS

**Listing 1**

```
public void init(DeploymentInfo di) throws DeploymentException {
    try {
        if (di.watch == null) {
            // resolve the watch
            if (di.url.getProtocol().equals("file")) {
                File file = new File(di.url.getFile());
                // If not directory we watch the package
                if (!file.isDirectory()) {
                    di.watch = di.url;
                }
                // If directory we watch the xml files
            } else {
                di.watch = new URL(di.url, "META-INF/jboss-
spring.xml");
            }
        } else {
            // We watch the top only, no directory support
            di.watch = di.url;
        }
    } catch (Exception e) {
        log.error("failed to parse Spring context document: ", e);
        throw new DeploymentException(e);
    }
    super.init(di);
}

private URL getDocUrl(DeploymentInfo di, URLClassLoader localCL)
throws DeploymentException {
    URL docURL = di.localUrl;
    if (!di.isXML) {
        docURL = localCL.findResource("META-INF/jboss-spring.
xml");
    }
    // Validate that the descriptor was found
    if (docURL == null) {
        throw new DeploymentException("Failed to find META-INF/
jboss-spring.xml");
    }
    return docURL;
}
```

**Listing 2**

```
public void create(DeploymentInfo di) throws DeploymentException {
    try {
        beanFactoryLoader.create(di);
        Notification msg = new Notification("Spring Deploy", this,
getNextNotificationSe-
quenceNumber());
        sendNotification(msg);
        log.info("Deployed Spring: " + di.url);
    } catch (Exception e) {
        throw new DeploymentException(e);
    }
}

public void create(DeploymentInfo di) throws DeploymentException {
    URL docURL = getDocUrl(di);
    String defaultName = getDefaultName(di);
    BeanFactory beanFactory = null;
    ClassLoader cl = Thread.currentThread().getContextClassLoad-
er();
    try {
        Thread.currentThread().setContextClassLoader(di.ucl);
        beanFactory = createBeanFactory(defaultName, new
UrlResource(docURL));
    } finally {
        Thread.currentThread().setContextClassLoader(cl);
    }
    String name = ((Nameable)beanFactory).getName();
    bind(beanFactory, name);
    log.info("Bean factory [" + name + "] binded to local
JNDI.");
    beanFactoryNamesMap.put(docURL.toString(), name);
}
```

**Listing 3**

```
<beans>
<description>BeanFactory=(myBeanFactory) ParentBeanFactory=(pa
rentBeanFactory)</description>
...

protected void preProcessXml(Element root) throws BeanDefinitionS
toreException {
    NodeList nl = root.getChildNodes();
    for (int i = 0; i < nl.getLength(); i++) {
        Node node = nl.item(i);
        if (node instanceof Element) {
            Element ele = (Element) node;
            if (DESCRIPTION_ELEMENT.equals(node.getNodeName())) {
                String nodeValue = ele.getFirstChild().getNode-
Value();
                log.info("Bean names [description tag]: " + node-
Value());
                Matcher bfm = parse(nodeValue, BEAN_FACTORY_
ELEMENT);
                if (bfm.find()) {
                    beanFactoryName = bfm.group(1);
                }
                Matcher pbfm = parse(nodeValue, PARENT_BEAN_
FACTORY_ELEMENT);
                if (pbfm.find()) {
                    String parentBeanFactoryName = pbfm.group(1);
                    try {
                        beanFactory.setParentBeanFactory(Beans
ryLoaderImpl.lookup(parentBeanFactoryName));
                    } catch (Exception e) {
                        throw new BeanDefinitionStoreException("F
ailure during parent bean factory JNDI lookup: " + parentBeanFactory-
Name, e);
                    }
                }
            }
        }
    }
}
```

**Listing 4**

```
public void stop(DeploymentInfo di) throws DeploymentException {
    try {
        URL docURL = getDocUrl(di);
        String name = (String) beanFactoryNamesMap.remove(docURL.
toString());
        BeanFactory beanFactory = lookup(name);
        doClose(beanFactory);
        unbind(name);
        log.info("Bean factory [" + name + "] unbinded from local
JNDI.");
    } catch (Exception e) {
        throw new DeploymentException(e);
    }
}
```

**Listing 5**

```
public class SpringInjectionInterceptor extends SpringInjectionSuppo
rt
implements Interceptor {

    public String getName() {
        return "SpringInjectionInterceptor";
    }

    public Object invoke(Invocation invocation) throws Throwable {
        if (!(invocation instanceof ConstructorInvocation)) {
            throw new IllegalArgumentException(
                "This interceptor is meant to be applied" +
                " only on new instantiation of EJB objects");
        }
        Object target = invocation.invokeNext();
        inject(target);
        return target;
    }
}
```

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

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



**ONLY**  
**\$69<sup>99</sup>**  
ONE YEAR  
12 ISSUES

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

[www.JDJ.SYS-CON.com](http://www.JDJ.SYS-CON.com)  
or **1-888-303-5282**



OFFER SUBJECT TO CHANGE WITHOUT NOTICE



# Developing Web Services

## Eclipse Web Tools Project

*Both bottom-up and top-down*

by Boris Minkin

Today's trend is to integrate existing systems in a standard way to make disparate implementations interoperate. Web Services and XML came along with the ability to provide a standard communication interface between these systems, as well as the standard description language – WSDL – the Web Services Description Language that lets those systems define the structure of the services they're providing. Web Services are built using three classic components:

- **SOAP** – Simple Object Access Protocol – the XML-based communication protocol for sending data using Web Services.
- **WSDL** – Web Services Description Language – the XML-based language that describes the Web Service that's provided by a particular system and how to access it.
- **UDDI** – Universal Description Discovery Integration – a directory that helps to identify dynamically where particular Web Services are and how to find them, as well as a means of publishing services for those who want to provide them.



**Boris Minkin** is a Divisional Vice President of a major financial corporation. He has more than 12 years of experience working in various areas of information technology and financial services. Boris is currently pursuing his Masters degree at Stevens Institute of Technology, New Jersey. His professional interests are in the Internet technology, service-oriented architecture, enterprise application architecture, multi-platform distributed applications, and relational database design.

bm@panix.com

There's also a wide variety of complementary standards that deal with things such as security (WS-Security), policy (WS-Policy), and reliability (WS-Reliability) to provide standard ways of ensuring secure and reliable communications through Web Services. However, these are beyond the scope of this article, which covers the basics of Web Services development using Eclipse Integrated Development Environment (IDE) and the Web Tools Project. Generally, developing Web Services in Java can be either bottom-up or top-down. In the bottom-up development mode, we start with an existing Java application and with either a Java bean or Stateless Session EJB. According to Java Web Services standards (JSR 101/109), only such Java entities can be used as a source of possible Web Services. The IDE helps us generate WSDL from these entities, as well as service wrappers and locators ensuring easy access to them from inside the application code. With the top-down approach, those wrappers and service locators get generated from an existing WSDL file.

Software prerequisites before one can start using Web Services with Eclipse include:

1. J2SE 5.0 JRE: <http://java.sun.com/j2se>
2. Eclipse 3.1.1: <http://www.eclipse.org>

3. Eclipse Web Tools Project (WTP) 1.0: <http://www.eclipse.org/Webtools> (this can also be automatically downloaded and installed into Eclipse using its Help – Software Updates menu). When this article was written, this was the latest WTP release (released on December 23, 2005).
4. Tomcat 5.0: <http://jakarta.apache.org/tomcat/> (must be already installed and configured in Eclipse – if not, make sure to do this by going to the Eclipse Window menu, selecting Preferences, Server, Installed Runtimes, and then using Add button, select Tomcat 5.0 runtime, where you have to specify the location of the Tomcat server installation on your computer). Please refer to the article on creating Web applications with WTP for more detailed instructions on configuring Tomcat in Eclipse WTP: <http://java.sys-con.com/read/152270.htm>.

### Building a Bottom-Up Web Service

Suppose we want to build a Web Service that will give us the stock price quotes – including the latest price, volume, and trade date/time. For purposes of this demonstration, we'll use a simple application providing this service. Please

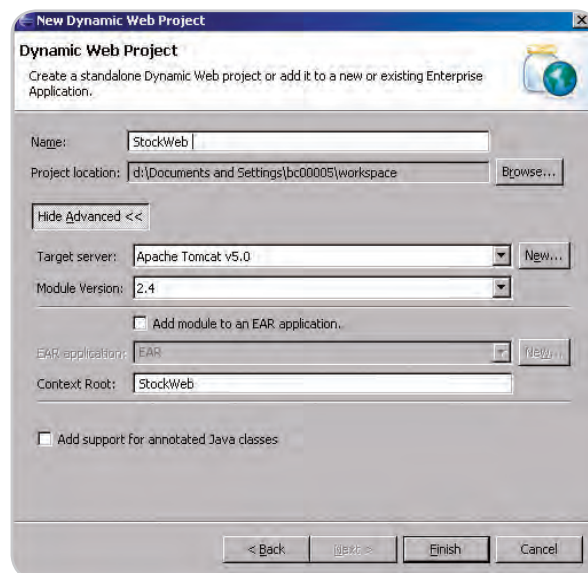


Figure 1 The wizard for creating Dynamic Web Project in the Eclipse WTP

# ENGAGE AND EXPLORE...

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

**JUNE 5-6, 2006** Roosevelt Hotel / New York, NY

## RESISTER TODAY!

### SOA 10th International WebServices Edge conference+expo



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

## RESISTER TODAY!

### 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](http://events.sys-con.com)**

Sponsored by

WebServices  
JOURNAL

XML JOURNAL

NET JOURNAL

eclipse developer's journal

WebSphere  
JOURNAL

Information  
STORAGE+SECURITY  
JOURNAL

wild THE OPEN SOURCE  
MAGAZINE

JDJ

LinuxWorld  
MAGAZINE

MX  
developer's journal

asp.netPRO

SDTimes

CoDe

Software Test  
& Performance

\*Call for Papers email: [events@sys-con.com](mailto: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

© 2005 WEB SERVICES EDGE. ALL RIGHTS RESERVED



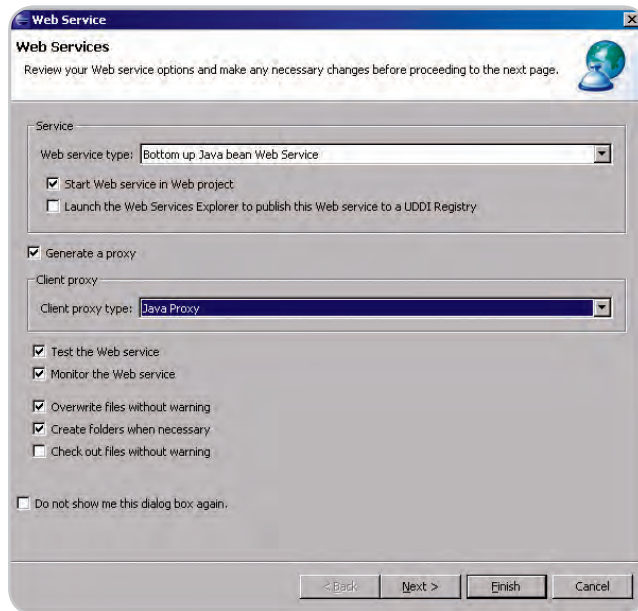


Figure 2 Creating a Bottom-Up Web Service wizard, initial screen

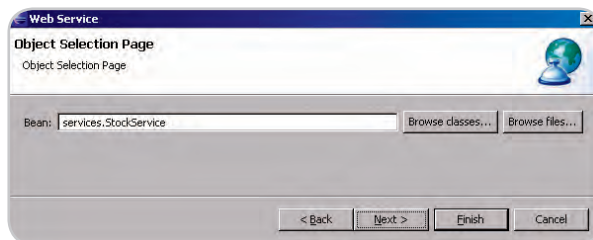


Figure 3 Selecting a Java Bean object to be exposed as a service

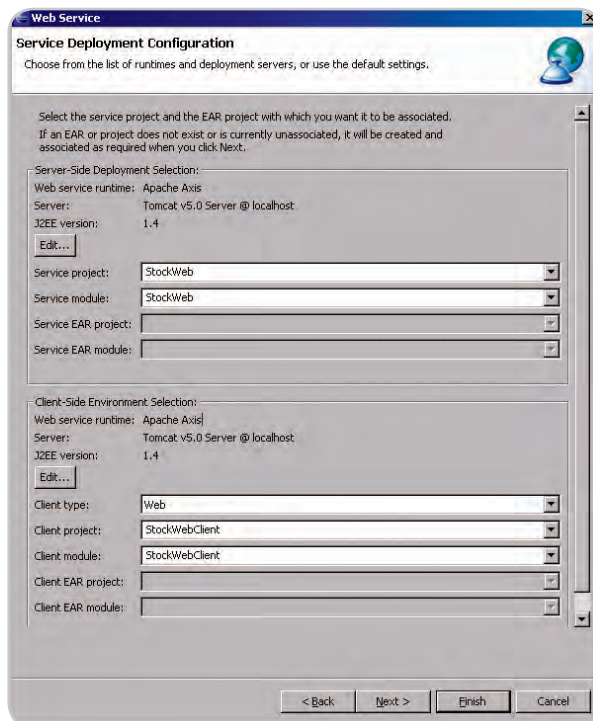


Figure 4 Selecting a service deployment configuration such as a server runtime, J2EE version, and Web Services engine runtime

note that this application adheres to Java Bean standards and implements appropriate getter and setter methods as well as the no-argument constructor.

Our simple application will consist of two classes: StockData and StockService. This application will provide information about stock price, volume, and trade date/time for a single item that we'll call "Our Stock" with the symbol OURS. The StockData class is a simple Java Bean that holds the stock's information. StockService holds the stock name and symbol and the history of its trading (we generate it randomly). These are the public methods that are available in the StockService class:

```
public double getLatestPrice() - returns the latest price of Our Stock
public long getLatestVolume(); - returns the latest volume number
public java.util.Date getLatestDateTime(); - returns the latest trade date
public String getStockName(); - returns stock name (preset to Our Stock)
public void setStockName(String nm); - sets the stock name to nm
public String getStockSymbol(); - returns stock symbol (preset to OURS)
public void setStockSymbol(String sb); - sets the stock symbol to sb
public void setLatestStockData(double price, long volume); -sets the latest stock data (adds another stock data object into our history)
public String getStockHistory(); - returns history of stock trading for Our Stock
```

Only public Java methods can be invoked using Web Services. To avoid platform interoperability issues and to be compliant with the JAX-RPC standard of simple-type encoding, we keep the method return parameters and arguments to simple Java types. The complete code of our StockData and StockService classes will be available with the article's source code (Please see <http://jdj.sys-con.com>).

Let's start building a Web Service from our StockService Java Bean. Please note that a similar procedure can be followed for an EJB Web Service, except in that case we have to build a stateless Session EJB first. Also, in the case of an EJB Web Service, an EJB project would have to be created in Eclipse, rather than the so-called Dynamic Web Project that we're building now.

First, we create a Dynamic Web Project called StockWeb. We'll use the Eclipse Web Tools wizard to create the Dynamic Web Project by selecting File-New-Other from the menu and then expand Web to Dynamic Web Project, which will bring up the screen depicted in Figure 1.

Press Finish to create the StockWeb project. Once created, we'll import our StockData.java and StockService.java files into the JavaSource directory. To import, simply highlight "JavaSource" and select File-Import from the menu and specify "File System" as a type of import. The wizard that appears allows the developer to specify the location of the files to be imported. Once we have these files imported (an appropriate package called "services" will be automatically created), we'll simply highlight StockService class, right-click, and select Web Services-Create Web Service from the context menu.



Figure 2 shows several important notations:

- **Web Service Type:** Bottom-up and Top-down – we already went through what those mean – since we’re creating a Web Service from an existing Java Bean, it’s a Bottom-up Web Service, so this selection will be automatically highlighted.
- **Start Web Service in Web project** makes sure that the Web Service is started when the wizard finishes creating it. Basically this ensures that when the Web Service creation wizard finishes, the Web Service will be automatically started along the Web project that contains it.
- **Generate a Proxy:** the only proxy type here is Java Proxy. This helps create a static proxy class – a Java bean class that would simplify invoking this Web Service from within the application. It would actually act as a Web Service client.
- **Test the Web Service:** specifies whether we want to automatically launch the associated Tomcat runtime to launch the Web project along with the Web Service and see if it works.
- **Monitor the Web Service:** Eclipse Web Tools include a built-in monitor that helps monitor SOAP envelopes –XML-based definitions of communication chunks flowing back and forth between the client and service.
- **Overwrite files without warning** and **Create folders when necessary:** they’re pretty much self-explanatory.
- **Check out files without warning:** is for use with source code management (SCM) systems such as PVCS, CVS, or ClearCase – file are checked out (locked for editing by a particular developer) automatically.

Click the Next button. This will bring the simple screen portrayed in Figure 3 that shows the Java Bean we’re building the Web Service from. Since we highlighted StockService class when we started this process, the wizard will automatically show it.

The next screen is very important; it configures the service’s deployment. Since we’ve elected to generate a Web Service proxy (a client for our Web Service), a new Web project will be dynamically generated for us that contains all the classes for accessing and invoking our Web Services – our Web Services client.

On this page, we can also select the J2EE version, Web Services runtime engine, and server where the Web Service will be deployed. We highly recommend selecting J2EE 1.4, and no lower, since with J2EE 1.4, Web Services became a part of the Java 2 Enterprise Edition platform, while previous versions treated them as a specific add-on. The Web Tools Project doesn’t fully utilize this feature of J2EE yet; however, relying on it will position you better for future releases. Also, the only engine that the Web Tools Project currently supports is the Apache Axis engine. It’s a second attempt by Apache to implement Web Services standards. The first attempt, called Apache SOAP, had some deficiencies, especially, major performance issues. Apache Axis seems to overcome these issues by using SAX parsers more often than the DOM ones. A new Web Project – StockWebClient – will be created for us to host the client Web application for our Web Service.

In Figure 5 we’ll have a chance to select which methods in the StockService class should be exposed as Web Services. On this screen we’ll also have a chance to select our Encoding

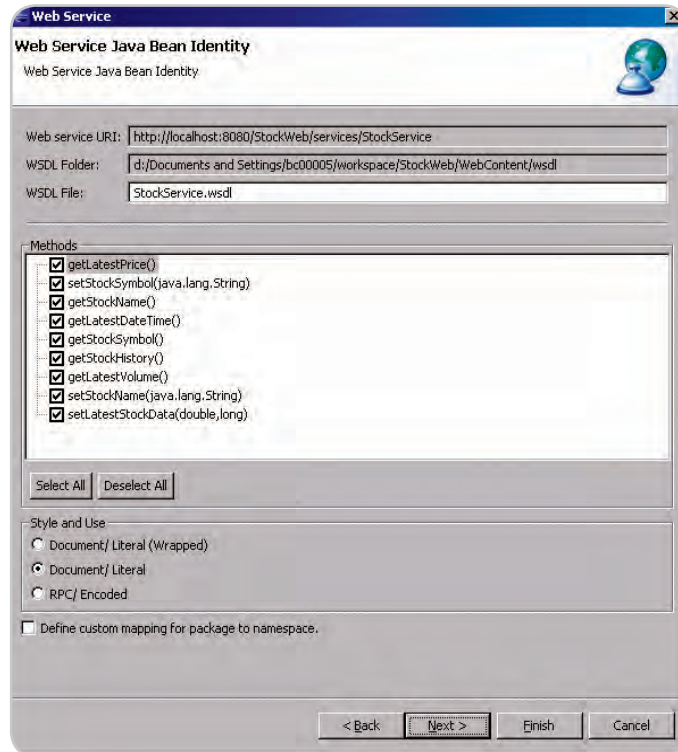


Figure 5 Select the Web Service encoding style as well as the select methods that should be exposed by a Web Service

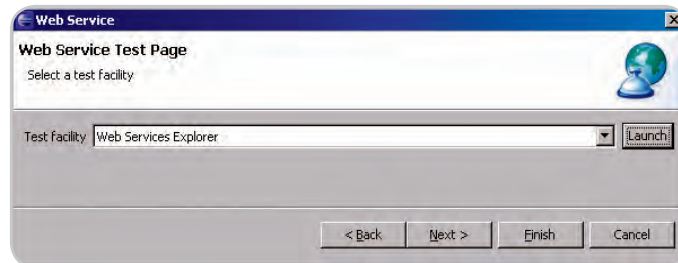


Figure 6 Web Service Testing facility selection

scheme: Document/Literal versus RPC/Encoded. Generally speaking, Document/Literal is much more common and provides much greater interoperability between platforms, though RPC might provide somewhat better performance, so it may be considered for homogeneous platforms with no interoperability requirements. We’ll select Document/Literal for greater interoperability; in our example we aren’t particularly concerned about performance.

Please note that the screen in Figure 5 also designates the service URI and the location where the WSDL file will be generated. Once we click “Next” and pass this screen, the Web Service will be generated and the server will be launched so we can test this Web Service.

The screen in Figure 6 specifies the testing facility for our Web Service, which is Web Services Explorer – it allows browsing and testing Web Services, invoking individual methods, and getting results. Click Finish to finish generating our Web Service. Once the service is started, Web Services Explorer is automatically launched, and we can test our Web Service as shown in Figure 7.

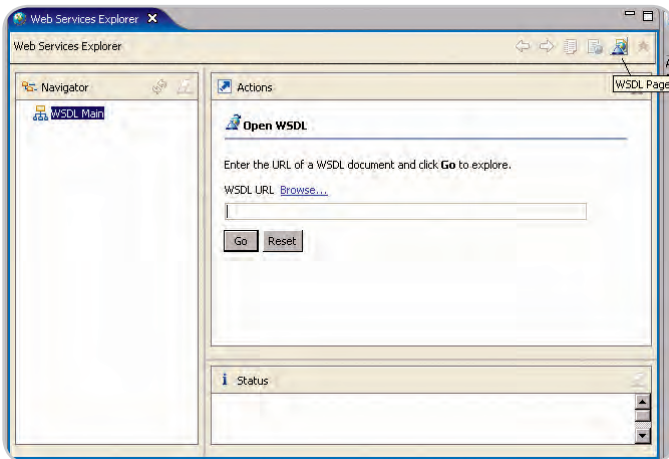


Figure 7 The Web Services Explorer WSDL page lets you select a Web Service for testing in the Eclipse workbench

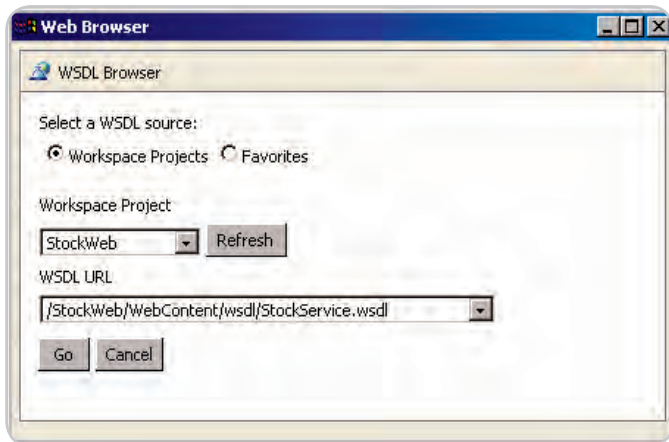


Figure 8 Selecting Web Services in the Workspace Project

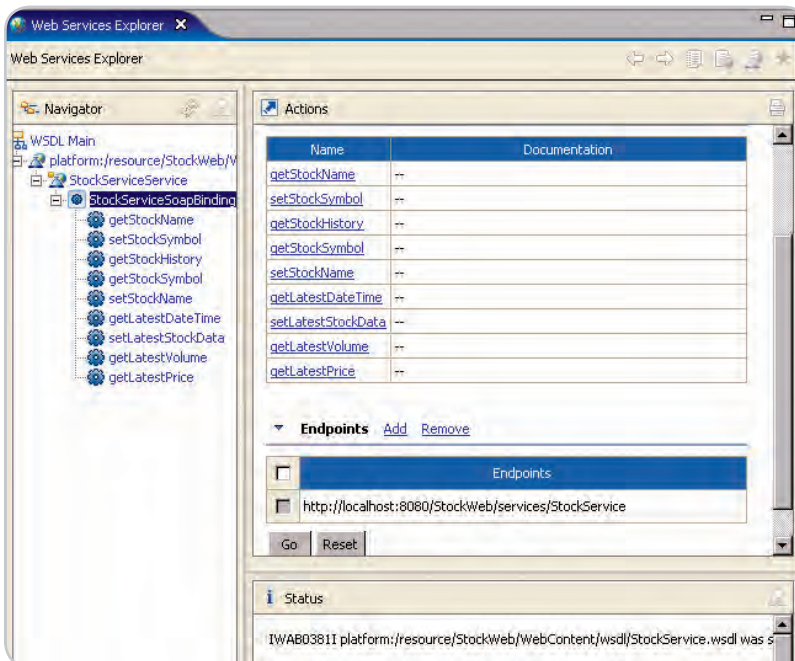


Figure 9 Back to Web Services Explorer – with the Web Service (to be tested) selected

Select the WSDL page icon (highlighted in the screen in Figure 7) in the Web Services Explorer then click WSDL Main in the right pane; this gives us the opportunity to invoke the Web Service directly by pointing to its WSDL. When you click “Browse,” the screen shown in Figure 8 appears.

On this screen, a developer can select a project from an Eclipse Workspace then select the WSDL URL available in this project, in our case StockService.wsdl. Pressing Go and then Go again brings us back to the WSDL Explorer screen (see Figure 9), where all the methods exposed as part of our Web Service interface are generated:

You can specify the parameters (if any are required) and invoke this method as necessary by clicking on a particular method.

Now let's take a look at the entities that were generated. First of all, it's the WSDL file, StockService.wsdl that gets automatically generated under the Web Content\wsdl directory. Eclipse provides a superior editor for editing WSDL files as shown in Figure 10.

It includes the following four major interrelated entities that describe every Web Service:

- **Types:** A container for data type definitions using XML schema type system.
- **Messages:** An abstract typed definition of the data being communicated. A message can have one or more typed parts, for example, the highlighted message getLatestDateTimeResponse has just one part that is its return parameter of xsd:date (XML Schema date type).
- **Port types:** Abstract sets of one or more operations supported by one or more ports.
- **Operations:** Abstract description of an action supported by the service that defines the input and output message as well as optional fault message.
- **Bindings:** Concrete protocol and data format specification for a particular port type. The binding information contains the protocol name, the invocation style, a service ID, and the encoding for each operation.
- **Services:** A collection of related ports.
- **Ports:** Single endpoints, which are defined as an aggregation of the binding and network addresses.

Also, the Web Services client project StockWebClient got generated, where we have the following four classes:

- **StockServiceServiceLocator:** the locator class has information about the Web Service, most importantly the address by which the service can be invoked (the HTTP address). The locator class is populated based on the content of the Web Service's WSDL file.
- **StockService:** the service endpoint interface that is a local representation of a Web Service. Through this interface, appropriate methods will be invoked.
- **StockServiceSoapBindingStub:** the stub that implements remote invocation methods defined by StockService interface.
- **StockServiceService:** the service locator interface that defines methods implemented by service locator class.

To invoke our service from the client, we can construct the following simple Java application under our StockWebClient Web project as shown in Listing 1.

Invoking this client can be done simply through the command line or using the Eclipse: right-button menu, Run-As, Java Application. This demonstrates the remote communication that Web Services provide – we invoke a service that could be located on a different machine so long as we know the endpoint address of the service as defined in the StockServiceLocator. Here's the output of this class:

```
Stock name: Our Stock
Stock price: 30.63180066603478
Stock volume: 42475629656767
```

## Building a Top-Down Web Service

The top-down approach is commonly used when we have a standard service definition and want to implement this definition to provide the requested service. Top-down Web Service design involves the following steps:

- First, we locate the WSDL document (either someone gives it to us, or sends it to us via email, or we locate it in Web Services Inspection Language (WSIL) document. Web Services Inspection Language (WSIL) allows one to describe multiple WSDL documents located within a site. We can also search UDDI registry for our Web Service.
- Next, we generate implementation skeleton (normally, a Java bean skeleton), which contains methods and parameters that we fill to implement our Web Service.
- Finally, using this skeleton, we complete all the methods with the appropriate logic.

After highlighting our StockService.wsdl document under Web Content\WSDL, we can right-click and select the Web Services - Generate Java Bean Skeleton from the right-button context menu. This brings up the dialog shown in Figure 11.

It's very similar to the one we initially used in the Bottom-Up development section, when we clicked "Next"; it brings us to the screen where we can specify the location of the WSDL file as shown in Figure 12.

The next screen (see Figure 13) shows the service deployment configuration: which Web project, server, Web Services runtime, and J2EE version to deploy the new Web Service to. Note that we may want to select a different Web project (say StockWebOne) here so as not to overwrite our sample classes from the previous example.

Click Finish to generate a bunch of classes including the service endpoint interface, service locator, but also the most interesting of them, the StockServiceSoapBindingImpl class, where method stubs are provided for you to fill out the particular implementation of this Web Service as shown in Listing 2. (Listing 2 and additional source code can be downloaded from <http://jdj.sys-con.com>.)

A developer has to fill out these method stubs with the appropriate logic – obviously the system can't do it automatically for him since the WSDL file that the Web Service is generated from in a top-down approach only has information about Web Service's interface, not the actual implementation.

## Testing and Debugging Web Services

Testing and debugging Web Services is done using Web Services Explorer. It has three major panes: WSDL, WSIL, and

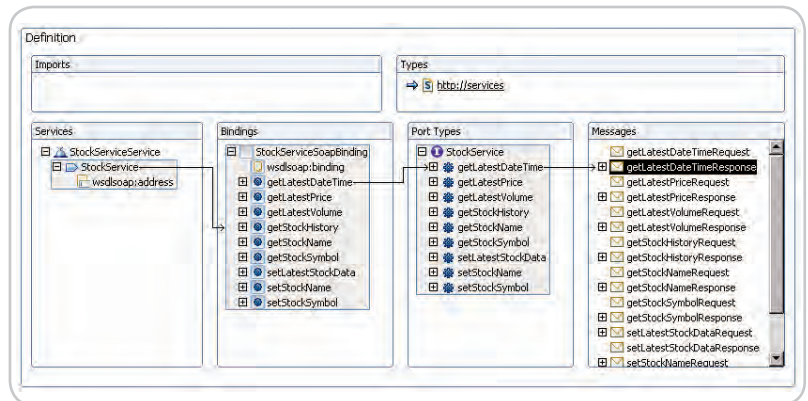


Figure 10 WSDL file editor

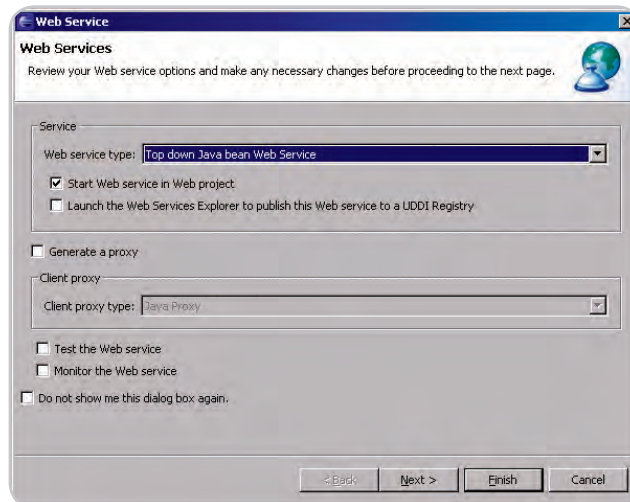


Figure 11 Wizard dialog for creating a Top-down Web Service

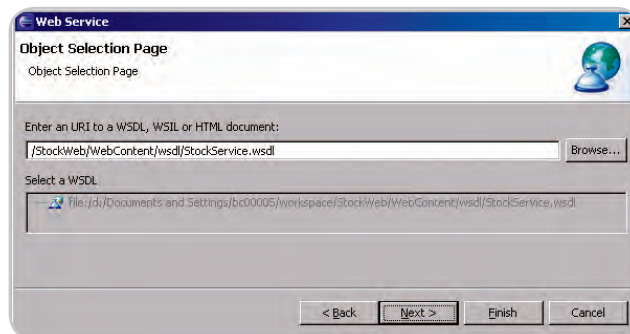


Figure 12 Select a location for the WSDL file to generate a Top-down Web Service

UDDI. Each of these serves the purpose of finding and testing particular Web Services. In the WSDL pane you can do it directly by finding the WSDL document in your project. The WSIL pane makes it easier for a developer to locate and test a particular Web Service on a particular site. With the help of the UDDI interface, one can search private or public UDDI registries (a private registry usually belongs to a particular company and its intranet services, which are usually invoked by internal applications at that company. Public registries are available throughout the Internet and are hosted by companies such as Microsoft [see <http://uddi.microsoft.com/>] for public use.)



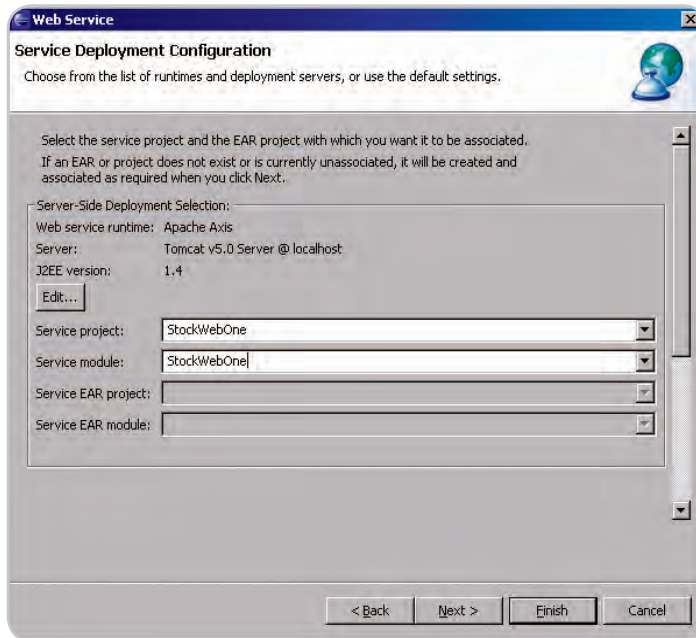


Figure 13 The service deployment configuration for our top-down Web Service

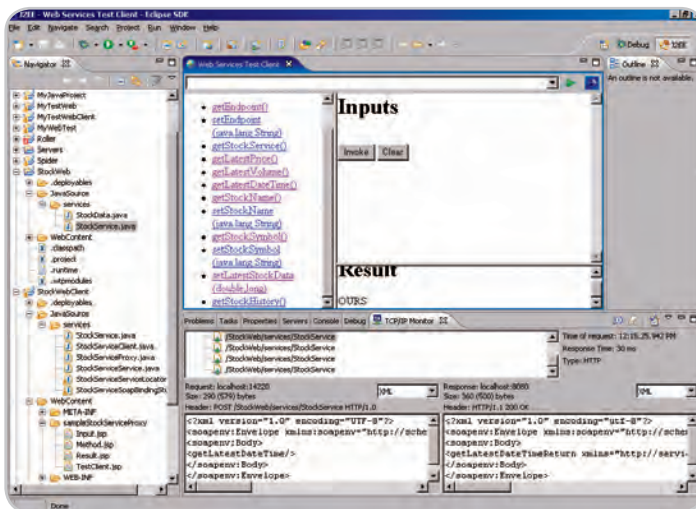


Figure 14 Testing and debugging Web Service with the help of the TCP/IP monitor

Another way to test Web Services is by generating sample JSP files that create a sample application communicating with the Web Service. This can be easily done by right-clicking on the Java Bean and selecting the Web Services – Generate Sample Pages after Web Service creation is finished.

You can also generate a client from a WSDL file. It will include the service locator, service binding stub, and service endpoint interface: right-click and select Web Services – Generate Client. This will let you construct a simple Java application using the generated classes, just as we did in top-down Web Services generation.

All Eclipse debugging facilities are available when testing Web Services. A developer can start servers in the debugging mode and set breakpoints at any line to do normal code debugging.

Eclipse also includes a TCP/IP monitor to watch the Web Service methods' executions. Figure 14 shows how the TCP/IP

monitor lets you debug a Web Service by using a client that consists of generated sample JSP pages. The TCP/IP monitor contains three panes:

- The right-top pane includes a list of the interactions (requests/responses) that have been done in chronological order. When a developer clicks on a particular interaction, the system displays the time of request, how long it took the system to respond (in milliseconds), and the type of protocol that was used (in our case, HTTP).
- The left-bottom pane displays the contents of the SOAP envelope generated by the Web Service's request.
- The right-bottom pane displays the SOAP response envelope. Whenever we invoke a particular Web Service, this pane is populated with the response of the server in the SOAP (XML-based) format.

A TCP/IP monitor is a powerful facility showing the data that are actually being sent through the wire and simplifies analyzing possible problems with the Web Services implementation. If a developer uses one of the Web Services Security standards such as encryption, it would also allow him or her to see whether the request and response were properly encrypted and are resilient to possible intruder attack. However, currently Eclipse WTP tooling doesn't support the Web Services Security standard; a developer has to manually refer to this facility if he or she is to leverage it.

## Conclusion

In this article, we've shown you how to develop bottom-up and top-down sample Web Services using the Eclipse Web Tools Project. Future articles will cover developing database-driven Web Services, security issues, and interoperability between Java and .NET.

### Listing 1: Creating a simple Java client to invoke a Web Service through the generated service locator and service endpoint interface

```
package services;

/**
 * To invoke stock service
 */
public class StockServiceClient {
    /**
     * To invoke stock service
     */
    public static void main(String[] args) {
        try{
            StockServiceServiceLocator wsl = new StockServiceServiceLocator();
            StockService ws = (StockService) wsl.getStockService();
            String name = ws.getStockName();
            System.out.println("Stock name: " + name);
            double price = ws.getLatestPrice();
            System.out.println("Stock price: " + price);
            long volume = ws.getLatestVolume();
            System.out.println("Stock volume: " + volume);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Advertiser	URL	Phone	Page
AJAX Seminar	www.ajaxseminar.com	201-802-3022	59
Altova	www.altova.com	978-816-1600	4
ArcMind, Inc.	www.arc-mind.com	520-290-6588	23
ceTe Software	www.dynamicpdf.com	800-631-5006	43
Elastic Path	www.elasticpath.com/flexible	800-942-5282	21
Extentech	www.extentech.com/jdj	415-759-5292	29
InterSystems	www.intersystems.com/cache16p	617-621-0600	7
IT Solutions Guide	www.itsolutions.sys-con.com	888-303-5282	61
ITVcon.com Conference & Expo	itvcon.com	201-802-3023	35
Java Developer's Journal	www.jdj.sys-con.com	888-303-5282	53
LinuxWorld Conference & Expo	www.linuxworldexpo.com/boston	800-657-1474	37
Northwoods Software Corp.	www.nwoods.com	800-434-9820	45
OpenAccess Software, Inc.	www.openaccesssoftware.com	888-805-ODBC	41
Parasoft Corporation	www.parasoft.com/jdjmagazine	888-305-0041	Cover IV
Quest Software	www.quest.com/g11	949-754-8000	Cover II
Software FX	www.softwarefx.com	800-392-4278	Cover III
SYS-CON Events	events.sys-con.com	201-802-3023	55
Synaptris	www.intelliview.com/jdj	1-866-991VIEW	15
TIBCO Software Inc.	www.tibco.com/mk/gi	800-420-8450	11
WebAppCabaret	www.webappcabaret.com/jdj.jsp	866-256-7973	19
Windward Studios, Inc.	www.windwardreports.com	303-499-2544	9
Xenos	www.xenos.com/VAN	888-242-0695	31
Xyθος Software, Inc.	www.xythos.com	888-4XYTHOS	33

**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.

# Reach Over 100,000 Enterprise Development Managers & Decision Makers with...



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

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

## Get Listed as a Top 20\* Solutions Provider

**For Advertising Details  
Call 201 802-3021 Today!**

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



**Joe Winchester**  
Desktop Java Editor

# Where Are the High-Level Design Open Source Tools?

I have just finished reviewing the book *Open Source Development Tools for Java*, which provides excellent coverage of such topics as log4J, CVS, Ant, and JUnit. There is a chapter on UML tools though in which the author almost apologizes for the lack of good open source design tools. There is a plethora of projects on SourceForge.net from J2EE runtime frameworks to IDE plugins, yet there is almost nothing that encroaches upward into the arena of analysis and design tools.

One theory for this is that high-level design tools are the value-add that software vendors hold back from the open source community and sell in high-priced offerings. While this does occur to a certain extent, it can't be the real reason. Open source is a marketplace where useful things become free and commoditized, as witnessed by runtimes such as Tomcat and JBoss or tools such as Eclipse and NetBeans. An open source business model that certainly works is to give away a product and then charge for consultancy and help using it. Is it just a matter of time before a big vendor starts to give away modeling tools? They could benefit if their product became a de-facto standard for developers, potentially quashing the competition in the process and growing the marketplace for a specialized consultancy.

Another reason given by a colleague, whom I posed the question to, was that design tools just aren't that useful. His measure of worth is a product that creates something he can compile, touch, execute, and debug. There will always be those who subscribe to the opinion that high-level modeling is the realm of bluff and fluffware practiced by those who masquerade their inability to write code behind its numerous charts and methodology steps. It's an unfair view though that can be equally leveled at developers who drown themselves in worrying about obtuse coding techniques instead of just writing a program that lets users get their job done more efficiently. The best tools are those that bridge the gap between the high- and low-level software disciplines

by seamlessly working with the same artifacts, presenting alternative views for disparate learning styles.

Visual learners prefer to think and work with charts and diagrams to analyze problems and communicate solutions, while I view coders as tactile or kinesthetic stylists who feel happiest with their IDE paused at a breakpoint showing them a stack trace from which they can explore and learn. There is a generation of tools that tended to be one way, where design charts generated code to be compiled and executed; these are probably the ones



my colleague spoke about so scathingly. These changes to the source code don't get reflected back in the tool and the code is undoubtedly more bloated and less efficient than if it had been written by hand. The developer who has to debug problems in the spaghetti unfairly stores his or her frustration as a general disdain for all design tools. Such attitudes are unfair and often dated, however, as there are some excellent design tools available that happily round-trip between high-level diagrams and actual code. A good example of this is Sun's *Visual Paradigm for Software Development Environment* <http://www.visual-paradigm.com/product/sde/nb/index.jsp> for NetBeans or Rational Software's *Rational Application Developer* <http://www-306.ibm.com/software/awdtools/developer/application/> that builds on an Eclipse codebase.

Returning to the plot line: Why then is the open source community starved of good, high-level tools? The reason I subscribe to is that the open source com-

munity just hasn't got around to creating them. The stack of open source software out there has been built from the bottom up, with small nimble runtimes and tight extensible IDEs. Such software is often built by those who use it themselves, providing a tight feedback loop between design and implementation that has resulted in the well-baked solutions that we take for granted as being freely available. For the most part, this space is well populated and there are a healthy number of offerings to choose from. I think that the future bodes well for a growth in open source tools that tackle and reach into the higher-level problem arena of software development. This could come about several ways. A tool such as ArgoUML (<http://argouml.tigris.org/>), a very solid open source UML product currently without an IDE home, might become more integrated with one of the major IDEs like NetBeans or Eclipse to piggyback a larger user base. The Eclipse Foundation has a tools project, Graphical Modeling Framework <http://www.eclipse.org/gmf/>, although this currently seems very focused on the Eclipse Modeling Framework as its high-level runtime rather than UML in general.

Another possibility is that a major vendor with a track record in open source tooling throws its product into the ring, or perhaps Borland will rediscover the mind-share it used to have with TogetherJ in one of its designer or architect products.

Whatever occurs with design tools and open source, I hope it's one that marries the best ideas from those with the knowledge of how a good high-level design tool should work with those who know how to code, implement, and manage a successful open source project. The outcome will be that we have a more productive suite of software development tools to choose from. Once this progression is complete, the problem arena will move even higher to open source tools that allow the end users to capture requirements in a form that communicates between their problem domain and the code. The sky's the limit. ☼

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

[joewinchester@sys-con.com](mailto:joewinchester@sys-con.com)



**SoftwareFX**

Bringing Your Data, Business & Strategy Into Focus.

**New!**  
**version 6.2**  
**Now Includes Maps!**

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

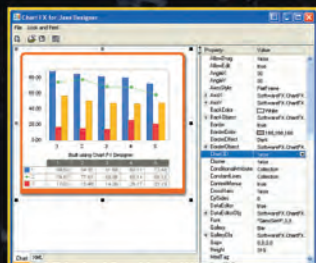


Chart FX for Java Designer

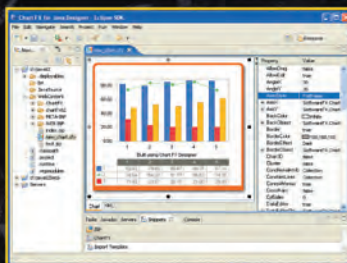


Chart FX for Java - Eclipse

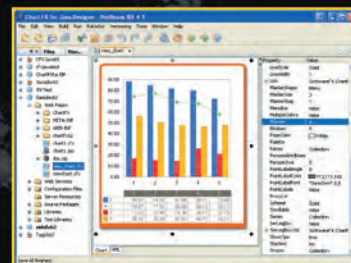
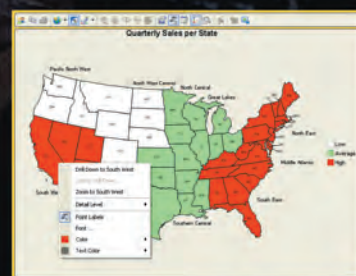


Chart FX for Java - NetBeans

## 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](http://www.softwarefx.com)

©2005 Software FX. All rights reserved. Chart FX is a registered trademark of Software FX, Inc. All other brands are owned by their respective owners.



# Code in quality. Test out bugs.

Take control of your code through Automated Unit Testing and Code Analysis...

**PARASOFT®**  
*Jtest*

- Automatically analyzes your code, applying 500+ Java coding best practices that surface poor code constructs affecting reliability, security and performance.
- Automatically unit tests your Java code evaluating code behavior and exposing unexpected exceptions, boundary condition errors and memory leaks.
- Generates extendable, high coverage test cases in JUnit format that can be quickly turned into libraries of reusable test assets.
- 100's of Quick Fix options for fast, accurate resolution of errors.
- Integrated code coverage analysis ensures that your code is thoroughly tested.
- Team collaboration support allows you to define, distribute and enforce code quality standards across entire development teams.

## Stop bugs before they start:

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

The result? Cleaner, safer, more reliable and consistent code. Reduced code rework. Faster, more predictable release cycles. Reliable applications and happier, more productive end-users. **To try Jtest today, call 888-305-0041 (x-3501) or go to [www.parasoft.com/JDJmagazine](http://www.parasoft.com/JDJmagazine).**

Parasoft Jtest clients include: Lehman Bros., Cisco, Disney, Raytheon, ADP, Sabre Holdings, GE and more.

**PARASOFT®**  
We make software work.™

**Automated Error Prevention™**

Parasoft Corporation, 101 E. Huntington Dr., Monrovia, CA 91016. For information, call 888-305-0041 (x-3501). Copyright ©2006 Parasoft Corporation. All rights reserved.  
All Parasoft product names are trademarks or registered trademarks of Parasoft Corporation in the United States and other countries. All other marks are the property of their respective owners.