USING THE JAVA PERSISTENCE API (JPA) WITH SPRING 2.0 PAGE 18

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

JDJ

JDJ.SYS-CON.COM | VOL.12 ISSUE:4

USING THE FORCE [AND OPEN SOURCE]

Enable a Module-Based Approach TO CONSTRUCTING SERVICES

PLUS...

- Bridging the Gap Between Open Source and Commercial Applications
- Not Invented Here: Reject, Repulse, and Reinvent

Altova® MissionKit™ 2007 – Intelligent tools for XML developers and software architects.

ALTOVA® missionkit™ 2007

Gear up for development excellence

Take off with the Altova MissionKit, and discover the secret to savings on top software tools.

Spied in the Altova MissionKit 2007:

- The world's leading XML development tools
- Plus application design, data management, and modeling options for software architects

The Altova MissionKit 2007 bundles Altova's intelligent application development, data management, and modeling tools at 50% off their regular prices. Available in a variety of configurations tailored to the needs of software architects and XML developers, the Altova MissionKit delivers the highest functionality and best product value. It's your first-class ticket to the power, speed, and simplicity of Altova's award-winning product line. Save a bundle!

Download the Altova MissionKit 2007 today: www.altova.com

ALTOVA®

JAVA DEVELOPER'S JOURNAL

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

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

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

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

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

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

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

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

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

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

Not Invented Here: Reject, Repulse, and Reinvent

Joe Winchester

The phrase "not invented here," or NIH, when applied to technology, describes a resistance by a group to use a perfectly valid solution to a problem they're encountering because they'd rather build the answer from scratch than adopt something existing that already does the job. Assuming that there are no legal or licensing issues to stop the already-built technology from being included, the reasons behind the recalcitrance to its usage usually boil down to human nature.

Software engineers are inventors, and inventors like to build things from first principles. Arriving at their door with a completed solution takes the wind out of their sails; it undermines their relevance and forces them to examine something built by people who are possibly smarter than themselves. Most scientists revel in such group sharing of knowledge; as Sir Isaac Newton acknowledged, "If I have seen further it is by standing on ye shoulders of Giants." Only by recognizing, embracing, and then using the solution as a platform to further advance can science move forward. At a recent presentation given by some NASA engineers who base their command and control systems on the Eclipse Rich Client Platform, they described the decision to do this by drawing analogies to rockets. As developers, their job at NASA is to work on the payload of the rocket and perfect the portion that rides on the top to boldly go and do novel and exciting stuff in space. The lower part that propels them into orbit is basically a commodity to them and bought off the shelf from Boeing and other big companies whose expertise is in moving big and heavy things efficiently through the atmosphere. This embracing of Newton's wisdom to pre-requisite technology such as Java and the Eclipse RCP is not rocket science to mature and sensible developers.

Leaving the world of space exploration for a moment, one of Java's problems since its inception has been the model by which classes are loaded into a JVM. Simplistically speaking, when a JVM starts it is given a classpath that contains a list of directories, or .jar files, and each time the JVM wants to load a class it scans the classpath until a matching .class file is found. Having been located, the .class file gets loaded by the JVM's ClassLoader and instances can be created. This model of having a JVM start up with all of its classpath ducks lined in a row, requiring termination and restarting when anything changes, is overdue for an overhaul.

Fortunately, the OSGi alliance, www.osgi.org, has been tackling this

–continued on page 6

> "Software engineers are inventors, and inventors like to build things from first principles."

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

joewinchester@sys-con.com

Speed. Simplicity. Style.

Build better UIs with our JSF components.

NetAdvantage®
for JSF 2006 Volume 2

AJAX-enabled JavaServer™ Faces components

Simplify Complex Data – Our All-New Hierarchical Grid easily organizes and displays data in nested grids

Maintain Readability – Fixed Columns keep critical column data in view while your users scroll

Built-in Flexibility – Our APIs allow incredible interactive experiences on the web

Great User Experience – Our AJAX-enabled components turbo-charge your web applications for a rich client UI experience

learn more: infragistics.com/jsf

Infragistics Sales - 800 231 8588 Infragistics Europe Sales - +44 (0) 800 298 9055

Infragistics®
Powering The Presentation Layer

WINDOWS® FORMS ASP.NET WPF JSF

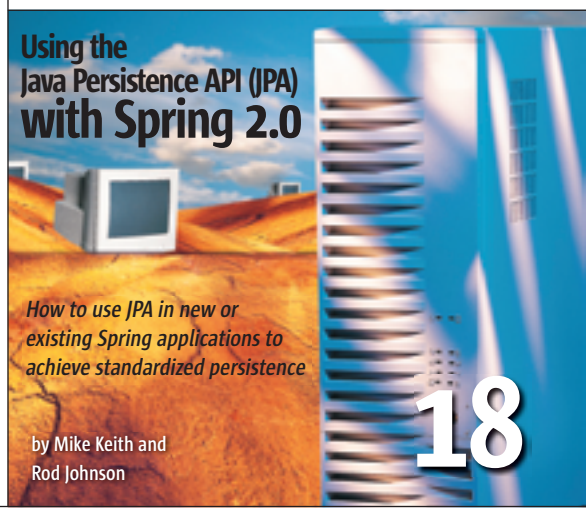
grids scheduling charting toolbars navigation menus listbars trees tabs explorer bars editors

APRIL 2007 VOLUME:12 ISSUE:4

JDJ contents

JDJ Cover Story

Feature



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

–continued from page 3

problem for about eight years now, during which time it has created an extremely mature and well-thought through dynamic module system for Java. OSGi's member companies are a who's who of today's software giants: BEA, Oracle, Motorola, IBM, Intel, Red Hat, Ericsson, and numerous others. Among the few absentees are Sun. The OSGi component software module model covers not only the execution environment, but encompasses life cycle management, as well as a registry of discoverable services. The technology is used today by the Eclipse framework and, along with the Standard Widget Toolkit, underpins applications built on the Rich Client Platform. A sensible strategy going forward would be for OSGi's Java component model to be included in the language. Such an initiative exists: Java Specification Request 291. JSR 291 was recently ratified by the JCP, a move that is goodness to all, from Java developers who can now build more flexible and robust applications, right through to the end users who will enjoy having more dynamic, more resilient, and more organic applications on their desktops, their servers, and their mobile devices. Everyone wins.

For some strange reason though, a cloud still lurks over JSR 291. Sun voted against ratification of JSR 291 in a move that many in the community are unable to reconcile as nothing more than an extreme case of NIH. In their defense Sun did have one other community member who voted with them to block the OSGi work becoming the de-facto Java module mechanism, Hani Suleiman, a gentleman not shy of sharing his opinions on the subject (http://www.eclipsezone.com/eclipse/forums/t92517.rhtml#92138332).

To complicate matters for JSR watchers, there is JSR 277 whose goal is to create a new static component model for Java 7. Many believe it is some kind of last ditch rearguard action to undermine JSR 291, OSGi, and, by ironic inference, the JCP.

Edmund Burke, the English philosopher, once remarked that, "The only thing necessary for the triumph of evil is for good men to do nothing." He was remarking about the fact that if nothing is done to counteract belligerence, arrogance, and bellicose behavior, then it will ultimately succeed. What I find remarkably refreshing about the Java space, however, is that time after time the recipe that triumphs for success is for strong positive arguments, strong technology, and strong communities to come together and tackle problems collectively. Those who choose not to participate are left behind and the consensus moves forward to a better place for the greater good of all involved.

As users of Java technology, we are all ultimately judged by the ability of the applications we build using the Java language to succeed or fail in front of our end users. The best way to serve them is to remain focused on their goals and wishes, collaborating with others if necessary to find answers to common problems, and intelligently assessing and adopting technology where relevant and applicable. The worst way to serve our end users is to adopt nihilistic attitudes to other's work, to engage in newsgroup insult wars over complex issues, or to attempt to apply a form of Java government that, like the ClassLoader, while relevant in the previous millennium, no longer pulls its weight today and is merely dated, anachronistic, and ineffective.

"As users of Java technology, we are all ultimately judged by the ability of the applications we build using the Java language to succeed or fail in front of our end users"

SYS-CON MEDIA

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

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

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

Advertising

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

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

Associate Sales Manager:
Corinna Melcon *corinna@sys-con.com*

Events

Events Manager:
Lauren Orsi *lauren@sys-con.com*

Events Associate:
Sharmonique Shade *sharmonique@sys-con.com*

Editorial

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

Production

Lead Designer:
Tami Lima *tami@sys-con.com*

Art Director:
Alex Botero *alex@sys-con.com*

Associate Art Directors:
Abraham Addo *abraham@sys-con.com*
Louis F. Cuffari *louis@sys-con.com*

Web Services

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

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

Accounting

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

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

Customer Relations

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

Experience JReport Live™

JReport® 8

Bring Your Reports to Life

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

Powerful Analytics Based on Dynamic Cube Technology

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

An Enterprise-class Foundation for Operational BI

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

Build Complex Precise Reports Quickly and Easily

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

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

Bring your Reports to Life with JReport Live Server
For more information, visit www.jinfonet.com/live

JReport®
JINFONET SOFTWARE

Bridging the Gap Between Open Source and Commercial Applications

Part 2 - Migrating EJB 2.0 entity beans to Hibernate POJOs

by Charles Lee

Migrating EJB 2.0 entity beans to Hibernate POJOs is pretty straightforward. Like many applications, all of the data for HQ is stored in the database, and we need to map from the underlying data store to an object-oriented view. In EJB 2.0, you would model that data with entity beans. An entity bean is created and found through the *Home* interface, and its fields are modified through its *Local/Remote* interface. These interface classes are automatically generated when we use XDoclet to annotate our entity bean implementations. We define the actual implementations in *EJBImpl classes. For example, for each getter/setter, we annotate the following:

```
/**
 * @ejb:interface-method
 * @ejb:persistent-field
 * @ejb:transaction type="SUPPORTS"
 */
public abstract String getName();
/**
 * @ejb:interface-method
 * @ejb:transaction type="MANDATORY"
 */
public abstract void setName(String name);
```

For all getters, we mark the transaction type as *SUPPORTS* so that it won't necessarily require a transaction. For all setters, the transaction type is *MANDATORY*, since modifications should be involved in a transaction. The same class file defines the create and finder methods, as well. The create method is a function body appropriately annotated with @ejb:create-method; and the finders are in EJBQL and written in pure annotation in this class file. In EJB 2.0, the concept of *Local/Remote* interfaces was introduced. We decided to go with only local interfaces for the entity beans so that they would not be accessible remotely, and thus bypass the permission checking that we do when they are accessed or modified. When we compile our code, out of each EJBImpl class, we automatically generate these additional classes:

- ***LocalHome*:** The local home interface that contains the create and finder methods
- ***Local:*** The local interface that contains the getter/setter methods
- ***Util:*** The utility class that fetches the home interface with the appropriate JNDI name

The N+1 Database Problem

The problem with entity beans has been termed the "N+1 database problem", and it's referring to the number of database calls to access entity beans. If a finder was invoked on the LocalHome interface, it will execute a query equivalent to the following:

```
select pk from table where column = value
```

PK is the primary key of the object, and most often it's the ID column. When you load each entity bean into memory by accessing its fields, the container will then issue another query:

```
select * from table where id = pk
```

If you do the math, for N rows found, you will end up issuing N + 1 queries to the database, henceforth the "N+1 database problem". Wait, there's more. We find that EJBs have a nasty habit of aggressively locking up database rows or tables no matter how much we tried to mark methods read-only, setting the container to lock optimistically, or even marking transaction type as *NOTSUPPORTED*. Maybe we didn't try hard enough, but we definitely pulled a fair amount of hair out over this. In fact, we found ourselves replacing getId() calls with getPrimaryKey().getId(), because while the ID has already been fetched into the primary key, the container would still do a database lookup when we ask for the ID. We needed to avoid the extra table lookups and reduce the duration of transactions to a minimum.

The Value Objects

XDoclet provides the facilities for implementing the Value Object pattern. The motivation for the pattern is to avoid calls through the entity bean's interface for each method call, which can be either local or remote. We took it a step further: we create a Value object (sometimes more than one so that we can have "light" objects that would incur the additional burden of loading up the relationships, etc.) for each entity bean and make sure to convert any lookups of entity beans to Value objects, and then cache the Value objects. Now, when the system is properly cached, we should be able to drop the "N" part of the "N+1" lookups. We augmented XDoclet a bit to account for this caching functionality. In our EJBImpl class, we also annotate the following for the generation of Value objects:

```
@ejb:value-object
      name="Resource"
      match="*"
      instantiation="eager"
      cacheable="true"
      cacheDuration="300000"
```

XDoclet will not only generate the Value object class, it will also generate an extension to our EJBImpl class to get/set its Value objects. Now this is starting to look a bit untidy. Take ResourceEBJImpl.java, for example; when we compile it, we get the following classes from one single file:

- ResourceUtil
- ResourceLocalHome
- ResourceLocal
- ResourceEJBImpl
- ResourceValue
- ResourceCMP

This simple task of mapping database tables to objects has quickly become cumbersome and error-prone. All these annotations and transaction problems can create bugs in areas that often cannot be detected until runtime.

Charles Lee is vice president of engineering at Hyperic.

Achieve Point-and-Click SOA™

exteria™

Business Automation Platform

Get on the SOA fast track!

NEXT STOP: extentech.com

FREE Exteria Downloads

Demos and Whitepapers

We're addicted!

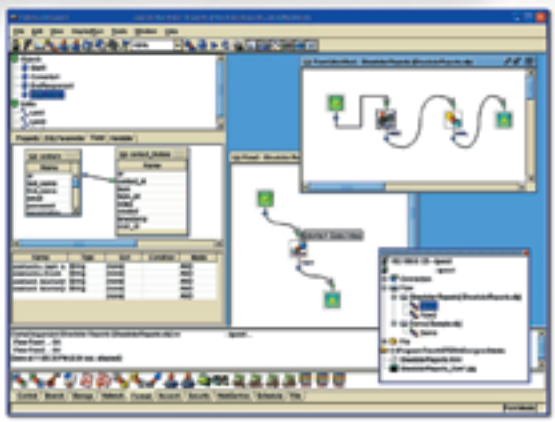
Now that we are going to output to the Web, ExtenXLS has saved us a tremendous amount of time and work."

-Steve T, System Architect, B2B Application Development Pfizer, Inc.

High-Speed Productivity

Exteria is the ideal modeling tool and web services platform for rapid SOA implementations.

Model data flows graphically in the code-free designer then publish with a click to XLS, PDF, SOAP, XML, and more -- all without writing any code!



The Exteria data flow modeler features dozens of connectors including LDAP, SOAP & XML

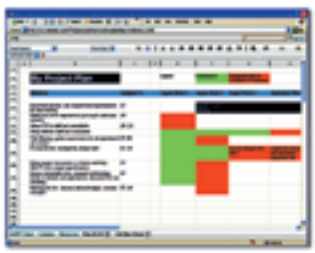
Spreadsheet Automation

ROI is all about re-use. There is no need to re-code the business logic already available in spreadsheets. Leverage this asset by mapping data flows to formula cells and publishing them as RSS feeds, REST API and SOAP web services, or to a web spreadsheet.

works with:

sheetster™

Integration with Sheetster.com web spreadsheets enables realtime data feeds and collaboration.



includes:

EXTENXLS 5™ JAVA SPREADSHEET TOOLKIT

Includes the best Java Spreadsheet Automation available... ExtenXLS 5.1!

extenTECH

exceed expectations

sales@extentech.com
call 415.759.5292
http://extentech.com/itsg

Moving On

Given the issues that we have had with the EJB 2.0 persistence layer, we decided to get rid of EJB 2.0 entity beans in favor of Hibernate. We would still keep the session beans, because they encapsulated our business logic. Hibernate uses plain old Java objects (POJOs) to map to the underlying data store. There is a mapping file (HBM) that maps the database columns to POJO fields and describes other object relationships. We had chosen to implement the Database Access Object (DAO) pattern, which abstracts and encapsulates all access to the data source. A DAO would define the create and finder methods, just like the entity bean Home interface, along with other CRUD methods. There would be a DAO that would be created for each Hibernate POJO to mirror the Home interface relationship. We set out with the following plan:

1. Create Hibernate POJO classes equivalent to all entity beans and the corresponding HBM files; in addition, the POJOs include getter/setter methods for the entity beans' Value objects.
2. Create corresponding DAO classes and migrate the finders from EJBQL to HQL (I will discuss the semantics of the finders and the conversion issues in a different article).
3. Replace all create and finder calls in session beans to use the DAOs.
4. Generate and commit Value object classes to the code base.
5. Remove the entity beans.

Pretty simple, right? In fact, it really was not more complicated than that. Granted, this plan was the second version from our exercise, but our application is divided up into subsystems. We were able to test out different strategies in one subsystem before applying the right one to the entire application.

The POJOs

First of all, creating the HBM files proved to be somewhat facilitated by the fact that we had our own syntax for defining database schema in XML, because we had developed our own database schema creation tool. We were able to go from something similar to our existing database schema files and apply an XSLT transformation to get the resulting HBM files that we needed with a bit of tweaking. The definition of container-managed relationships (CMRs) in the HBM files required some additional considerations, as they were previously annotated in the entity bean files. This is another topic that I will cover in a later piece. Now, some people choose to automate the generation of POJO files from the HBM files. Perhaps we were just a little tired of all the automatic code generation and decided to just hand roll the POJOs. It was not too much trouble, and we had more control over the content. There were other guidelines that we stuck to:

- ***No import of DAOs:*** POJOs should not be aware of any logic beyond the getter/setter of its properties.
- ***Setter methods are package scoped (default):*** We want to control the modification of data to only the classes in the same package as the POJOs, which are its DAOs, and the corresponding session beans that contain the business logic
- ***Collection getters return immutable collections:*** Since we wanted to encapsulate the modifiers to the package, we can't be returning modifiable collections through the public getter interface. However, we expose a separate packaged scoped getter method for the modifiable collection to the other classes in the package.

For example, for each operation, we associate a *Bag* of Roles:

```
public Collection getRoles() {
    return Collections.unmodifiableCollec-
tion(_roles);
}

Collection getRolesBag() {
    return _roles;
}

void setRolesBag(Collection val) {
    _roles = val;
}
```

We use a Pager concept extensively throughout the application to do paging, which would take a collection of entity beans and apply the appropriate paging and convert the collection to Value objects. The beauty of this scheme is that it provides a single point to apply the POJO to Value object transformation and achieve the same functionality as before. We wouldn't have to do too much more work after converting the application to use the DAOs' finders to convert to Value objects, thus preserving our APIs and caching behavior.

We used the Factory pattern to facilitate DAO instantiations. The DAOs encapsulated the CRUD and finder methods for the POJOs. Rewriting the finders in the DAOs provided some interesting challenges, but Hibernate Query Language (HQL) is much more capable than Enterprise JavaBeans Query Language (EJBQL). We did a first pass, converting everything from EJBQL to HQL, but then followed that with some conversions to criteria-based queries. The advantage of the queries inside actual function bodies in DAOs over the annotated finders in entity beans is the ability to debug and step through the query calls; through your IDE you can easily modify the pending query string on-the-fly to diagnose any problematic queries.

The biggest benefit from the conversion is the database-locking behavior. Hibernate is smarter about performing the write methods at the end of the transaction, instead of eagerly locking entity beans whenever their fields are being modified. We had to rid ourselves of all the *creative* transaction demarcation we were doing in HQ in order to reduce the involvement of transactions. Our previous strategy worked for us, but it was ugly. Besides the API calls being unpredictable whether it would be involved in a transaction, we were also sometimes creating extra transactions in any one given API call just to get nested transactions to close before returning. Because of Hibernate's behavior and transaction requirements, we had to go through and clean up all of the different transaction-type definitions and simply mark everything as REQUIRED. Believe me, not having to manage the complicated transaction demarcations was a big step forward for us.

This was the first phase in our conversion from entity beans to Hibernate, which lasted a little more than three weeks. We actually did not go for a gradual migration as we had originally intended; instead, we went all in. We converted roughly 80 entity beans to Hibernate. There were other issues and lessons in this conversion exercise, and we continued to evolve the code base from here. We replaced our homegrown database initialization routine, optimized queries, changed session behavior, added second-level caching to Hibernate, and have even begun to get rid of Value objects. However, this first step got us the eradication of EJB 2.0 entity beans, and it was good.

Oracle Fusion Middleware

Reduced Application Development Time by 50% With Oracle Fusion Middleware

Oracle Fusion Middleware

Hot-Pluggable. Comprehensive.

J2EE — Enterprise Portal — Identity Management — Integration — Data Hub — Business Intelligence

oracle.com/middleware
or call 1.800.ORACLE.1

Using the force (and open source)

Enable a Module-Based Approach to Constructing Services

by Franz Garsombke

The Jedi mind trick is a Force power that can influence the actions of weak-minded sentient beings. Vendors will often try to apply the Jedi mind trick in selling silver-bullet software solutions that solve global warming and stop celebrity feuding while enabling service-based architecture development. Let's quickly put on our aluminum foil caps and repel the Jedi mind trick by turning to open source solutions. Service-based architectures are being touted as the next step in reaching programming nirvana. With these marching orders it's often difficult to build a framework that allows for simple service creation. This framework should also be flexible, scalable, and lightweight as well as easy in exposing services externally. Without the correct framework(s) and guardrails in place your application services can quickly become a jumbled mess. Circular dependencies, massive Ant scripts, zero component reuse, and painful Web Service creation are just some of the problems faced by developers today. This article will demonstrate concepts on software lifecycle automation as well as a lightweight approach to creating a service-based architecture.

Franz Garsombke has been developing and architecting enterprise software solutions in Colorado for the last 11 years and is currently employed at Rally Software. He is a huge proponent of open source frameworks and passionate about developing and delivering simple quality pragmatic applications. He is proud to be the co-founder of a Java Bean mapping framework (http://dozer.sourceforge.net).

fgarsombke@yahoo.com

Building a Better Death Star

Everyone has worked on a software project delivered late (if at all) and over budget. I'm sure building the Death Star was no different. This article will demonstrate how to alleviate some of that pain by building a modular, reusable, and service-based architecture using a combination of open source projects. Maven, XFire, Spring, and Hibernate are all powerful frameworks when used alone. The real power is realized when they are used in conjunction. Maven is a software management tool that allows for module inheritance, dependency management, and overall project comprehension. XFire is a simple yet feature-rich framework that can expose any plain old Java object (POJO) as a Web Service. Spring is a layered Java application framework used to wire services together. Hibernate is an object-relational-mapping persistence and query service for Java.

You Say Jabba Desilijic Tiure, I Say Jabba the Hutt

Jabba Desilijic Tiure was one of the most notorious crime lords in the Star Wars galaxy. You probably didn't know who the term *Jabba Desilijic Tiure* referred to but if *Jabba the Hutt* was mentioned you'd know right away the creature in question. A common data dictionary is beneficial on any project or document. Before we continue a few key terms should be defined so it doesn't sound like this article was written in *Shyriiwook* (Wookiespeak).

- ***Service*:** The term service refers to a "discretely defined set of contiguous and autonomous business or technical functionality." One of the goals in creating a robust application framework is the ability to expose services through a number of different protocols. Creating services as POJOs will let us use other frameworks either to expose these externally or consume them internally.
- ***POM (Project Object Model)*:** The POM is an XML representation of a Maven project. A POM consists of information regarding dependencies, defect tracking, repository metadata, mailing lists, plug-in goals, and any other configuration details used to encompass fully the complete lifecycle of a project.
- ***Module*:** A module is a logical as well as physical part of your application. Layers and services in your application can be represented by physical Maven modules that logically make up the different components of your architecture.
- ***Build Dependencies*:** Maven manages different types of build dependencies for projects and cross-projects. A dependency is typically a jar file that a project needs for compilation, testing, or runtime behavior. Dependencies can be managed declaratively in your Maven project file. Maven also seamlessly manages transitive dependencies (dependencies of dependencies) and circular dependencies. Transitive dependencies are impossible to manage in Ant. Circular dependencies are particularly hard to manage in Ant since most of the time the source code for an application is compiled in one big mess. No feedback is given if, for example, the core layer of your architecture has a dependency on your service layer.
- ***Build Lifecycle*:** Software projects typically need the same tasks done to build and deploy code. Maven has a predefined set of phases each performing a series of actions to encompass a build lifecycle. Some common phases are validate, generate resources, compile, test,

package, integration test, and deploy. Maven plug-ins can be used to add functionality to a standard build lifecycle. Lifecycle phases should be managed by your software project management tool and not bolted onto a tool like Ant. Figure 1 shows some common lifecycle phases and the tasks they perform.

Just Say No to a 'Jabba the Hut' Architecture

It's more than likely that everyone has worked on an application that suffered from this syndrome. All it took was consuming one or two salacious crumbs a day without giving the application the care and feeding it needed. Bloating, circular dependencies, and the absence of separation of concerns are telltale signs that your architecture is turning into Jabba the Hut. Anyone using Java has surely written a few Ant files in the course of their career each consisting of thousands of lines of XML. Everything seems fine until your code base grows or more developers are added to the project. It's not simple to manage modules (or sub-projects) and dependencies using Ant as your build system. It's a great tool but it's outdated and too many people have tried to turn it into something it shouldn't be. A good analogy of Maven compared to Ant is the symbiotic relationship between Hibernate and JDBC. Hibernate provides an abstraction layer on top of JDBC and can provide a broader scope and range of features than the JDBC framework. The Maven plug-ins provide all the features of Ant as well as a standard way of interfacing with external tools and frameworks like Cargo (application container management), configuration management repositories, Hibernate, XFire, etc. Maven also brings a much broader set of features and concepts to the table than Ant.

Star Wars Convention? No, Not That Kind of Convention...

Maven's magic in building software is similar to the Ruby on Rails approach to programming in that most common tasks have default sensible strategies in place. This philosophy is called "Convention over Configuration." Trading flexibility at the infrastructure level by having standard directory layouts, standard naming conventions, and standard lifecycle phases promotes less emphasis on mundane details like building and deploying and allows for more time spent on the application's design and implementation. The guardrails Maven puts in place are necessary for a tight application but often don't get installed and are typically not given the care and feeding they need.

Software build structure layering should be the foundation of your application. This layering is prevalent at many levels in your ecosystem. It starts at your build system and feeds into your application. At an even broader scale this can be seen from your application into the enterprise. This was all made possible by a well structured build system.

I Am Your Father Luke

Project inheritance? Are you crazy? Wouldn't it be nice if there was a build system that handled software management lifecycle tasks and had some familiar and powerful object-oriented constructs like inheritance, encapsulation, cohesion, composition, and aggregation? I first used Maven 1 on an open source project solely for dependency management and site documentation. Frankly, I thought everything else it did was pretty miserable. I found myself always needing to drop into Ant to do trivial things. With the release of Maven 2 these highly desired features are present along with a lot more. This article is by no means a Maven tutorial but will show some key concepts as well as how to integrate it with some other important open source frameworks.

Maven comes with a feature rich set of plug-ins that provide core as well as enhanced functionality. Plug-ins are executed as part of the build lifecycle to perform tasks necessary to build a project. These plug-ins run unit tests, compile code, package software, and generate project reports among other things. Maven also has an extensible architecture that allows one to write any number of custom plug-ins to integrate with third-party tools or fulfill special tasks. One of the main benefits of using Maven is the ability to define a large amount of your build infrastructure in a parent project and inherit these dependencies and preconfigured plug-ins as well as any other lifecycle logic in your sub-modules. Here's a quote from *Better Builds with Maven* that sums up the tool, "Maven is a way of approaching a set of software as a collection of highly interdependent components that can be described in a common format. It's the next step in the evolution of how individuals and organizations collaborate to create software systems."

Now, Onto Building a Better Death Star

Everybody knows that the Death Star was destroyed by Luke Skywalker in *Star Wars Episode IV: A New Hope*. What they don't know is that another Death Star is being built using the latest and greatest Java frameworks. I guess you could say it's our lucky day. It's not easy constructing a monumental bringer of death out in the middle of nowhere. Droids, and many of them, must be gathered from across the galaxy to help in its construction. The system we'll be building for the Galactic Empire is an android (droid for short) provisioning system. I know that the Federation is in essence the Dark Side but the rates and benefits are outstanding. It doesn't hurt that there's free light saber training and laser target practice on Wednesday nights.

The key concepts that should be remembered throughout this exercise are:

- All of the services (modules) are POJOs that can be injected into any other module. These services could be a Web Service, data access service, common framework, business logic, or any number of other things required to make your application function.
- A service is a service is a service. Since our services are POJOs as well as Spring wired beans they can be exposed externally through any number of transport protocols. There are many frameworks that are tightly integrated with Spring that let you expose a Spring bean over HTTP, JMS, RMI, etc.

Typical Project Object Model (POM) Build Life Cycle

validate	compile	test	package	integration test	verify	install	deploy
validate all necessary information for the project	compile the source code	test the compiled source code	package up compiled source and other resources	deploy package for integration testing	verify package against criteria	install the package into the local repository	deploy the package to a remote repository

Figure 1 Common lifecycle phases

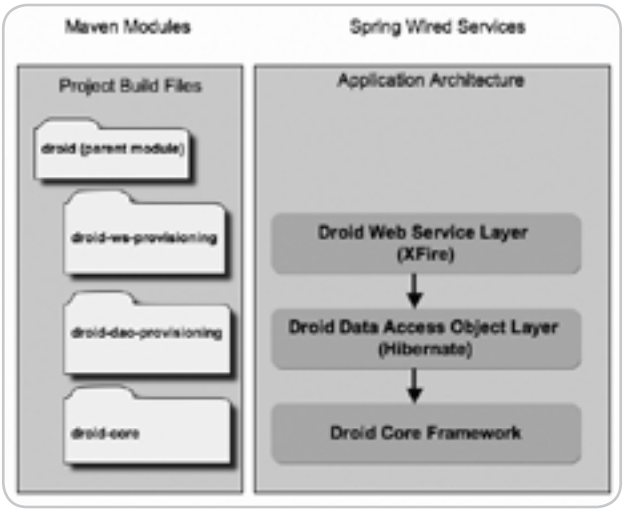


Figure 2 Visual representation of our physical project layout and its mapping to our logical architecture

Let's look at the services that comprise our droid provisioning system as well as the key technologies that are used to bring it to life. Figure 2 shows the Maven modules of the droid provisioning application and their mapping to logical services with technology implementations.

This sample application will demonstrate injecting services (both through Spring and Maven), WSDL-first development using XFire, and the creation of a data access service layer using Hibernate. All of the corresponding code for this example can be found at http://dozer.sourceforge.net/example.zip.

Anatomy of the Parent POM

Maven provides the concept of a parent module that is key to project inheritance. This module lets you define common dependencies, organizational information, deployment information, and common plug-ins. A skeleton of the provisioning application's POM file is shown in Listing 1.

This listing shows a typical POM layout structure. This file defines project-wide dependencies, project metadata, and common plug-in configurations.

In a Galaxy Far, Far Away

Let's code. The first thing we need is a common module that can be used by all upstream modules. This core module is comprised of one service but could potentially hold many others. Our core service really doesn't do much except elaborate on the concept of injecting and reusing services. Here's our service defined as a Spring bean in our module's Spring context file:

```
<bean id="coreService" class="com.examples.droid.core.service.
CoreService"/>
```

This bean references a Java class called *CoreService* that executes some core functionality.

The Maven configuration for this module is in Listing 2.

By defining our core module as a packaging type of *jar* Maven will compile our code, run unit tests, and package our classes into a jar file. The parent module inheritance is defined at the top of the configuration. No dependency version numbers are needed since they are defined in the parent. Let's see how this service can be used in sibling modules.

Springtime for Storm Troopers

Maven has the concept of dependency *management* to Spring's concept of dependency *injection*. At an abstract level the two are very similar and work hand-in-hand allowing for a reusable pluggable architecture. To add the core module as a dependency of another module we simply have to define it in our sibling module's POM file.

```
<dependencies>
    <dependency>
      <groupId>com.examples.droid</groupId>
      <artifactId>droid-core</artifactId>
    </dependency>
     …
<dependencies>
```

The version for the dependency has been defined in the parent module. To use the service defined in the core module we'd simply inject it into the service defined in the sibling module's Spring context file.

```
<bean id="provisioningServiceImpl" class="com.examples.droid.
ws.provisioning.ProvisioningServiceImpl">
    <property name="coreService" ref="coreService"/>
    …
</bean>
```

To reference the Spring context defined in the core module it needs to be included in the list of context files that make up the system's Spring *ApplicationContext*:

```
String[] configFiles = new String[] { "spring/droid-core-beans.xml",
                "spring/droid-dao-provisioning-beans.xml" };
BeanFactory factory = new ClassPathXmlApplicationContext(configFiles);
```

We now know how to create a service module and inject it through Maven dependency *management* as well as through Spring's dependency *injection*. Let's create some more services while working up our layers of the architecture.

Darth Vader Wants His TPS Reports!

Darth Vader has seen the initial budget for the new Death Star and is having heartburn. He needs a way to run some queries on our android provisioning database. Darth is pretty savvy so he chose Hibernate for the application's object relational mapping (ORM) persistence framework. Using Hibernate tools to reverse-engineer the database lets us dynamically generate our ORM files and data access objects. Since we rely on this generated code before we compile our persistence module we can simply bind the Hibernate tool's source generation task to a particular Maven lifecycle phase. The Hibernate source generation is done at the *generate-sources* lifecycle phase in Maven by defining our plug-in's phase element as shown in Listing 3.

This task was done in a *maven-antrun-plugin* as an Ant task since when this article was written the Hibernate tool's Maven plug-in wasn't as robust as the Ant task. The Maven variable *maven.compile.classpath* lets you have all of the module's dependencies and transitive dependencies at your fingertips.

Typically, a real-world application will have a database running for these Hibernate tools tasks to reverse-engineer against. The example code uses a HSQL database engine to simulate this behavior. The source code to start-up, create, and populate the database can be found in the example application.

The data access service is defined in our Spring context file in Listing 4.

The database consists of one table called *ANDROID* that's mapped to a Java class called *Android*. The generated data access object can then be used in our data access provisioning service:

```
public class ProvisioningDaoImpl extends AbstractDao implements
ProvisioningDao {

  public Android findDroid(String id) {
    return (Android) load(Android.class, id);
  }
}

public abstract class AbstractDao {

  public Object load(Class clazz, Serializable id) {
    return this.sessionFactory.getCurrentSession().load(clazz, id);
  }
}
```

The configuration needed for Hibernate is performed behind the scenes using Spring.

WSDL-First, You Must Do

Vader is getting impatient. He still can't access his TPS reports since there's no external API. We can expose our data access service with a Web Service using a simple framework called XFire. Let's examine the steps to enable simple top-down Web Service creation.

WSDL-first development is the concept of writing the interface for your Web Service before you write the code. WSDL-first development makes sense because it focuses less on a particular programming language and more on the messages between systems. Many people take the approach of generating the WSDL from existing code whether it's an EJB, POJO, or some other programming construct. WSDL-first takes a more top-down approach wherein the schema and WSDL are designed first and code generation happens from there. XML Schema is language-independent and provides more flexibility than a lot of programming languages. Client code, interfaces, implementation code, and schema types can all be generated from a WSDL and schema by a number of SOAP stack frameworks. One of the main benefits of WSDL-first development is improving interoperability between disparate systems programmed in different languages.

Taking our top-down approach let's look at an operation defined on our WSDL:

```
<wsdl:operation name="findDroid">
 <soap:operation style="document" soapAction="findDroid"/>
 <wsdl:input><soap:body use="literal"/></wsdl:input>
 <wsdl:output><soap:body use="literal"/></wsdl:output>
</wsdl:operation>
```

The *document/literal* approach to constructing a Web Service is more straightforward and solves many interoperability issues since it simply relies on XML Schema to describe exactly what the message looks like when delivered. Also, *document/literal* is WS-I (Web Services Interoperability) compliant. The request and response messages are defined below:

```
<xs:element name="findDroid" type="provisioning:findDroid"/>
<xs:complexType name="findDroid">
      …
</xs:complexType>
<xs:element name="findDroidResponse" type="provisioning:findDroid-
   Response"/>
<xs:complexType name="findDroidResponse">
      …
</xs:complexType>
```

By using a combination of XMLBeans (one of the many XML binding mechanisms supported by XFire), XFire, and our WSDL we can use separate tasks in Maven to generate everything needed to build a Web Service. There's an execution step defined in our *maven-antrun-plugin* in Listing 5.

All of the Web Service code generation is done in the *generate-sources* phase of the Maven build lifecycle. This lets us generate all of the messaging objects we rely on before our implementation code needs to compile against them. Since all of our messaging objects have been generated we can now implement our provisioning Web Service. XFire creates an implementation class based on the WSDL provided. That same class will be used to implement all of the business logic needed by our system.

The provisioning service is injected with the core service and data access service defined and implemented in earlier modules. The code in Listing 6 shows the provisioning service using the two services it's dependent on to look up android information.

The only thing left to do is wire up our provisioning Web Service. An important thing to note is that the provisioning service is a POJO. We could easily inject it into a different transport protocol implementation. XFire is used in this instance. We define our provisioning service as a service bean in XFire's service factory and are off and running.

```
<serviceFactory>org.codehaus.xfire.xmlbeans.XmlBeansServiceFactory</serviceFactory>
    <serviceBean>#provisioningServiceImpl</serviceBean>
  </service>

  <bean id="provisioningServiceImpl" class="com.examples.droid.ws.provisioning.
ProvisioningServiceImpl">
    <property name="coreService" ref="coreService"/>
    <property name="provisioningDao" ref="provisioningDaoImpl"/>
  </bean>
```

XFire takes care of the Web servlet, marshaling/unmarshaling of the SOAP messages, and determining which service and which operation needs to be executed. If our requirements dictate that we need a composite service comprised of the provisioning service and some other service we could just create a higher-level module and inject the services that have already been built. The provisioning service would then be just a POJO as opposed to a formal Web Service.

Empire Building

The example demonstrates how Maven can be leveraged to create an entire framework that developers can work in. If the infrastructure is set up properly developers can leverage and reuse many tasks through project inheritance and common plug-ins. Consistency goes a long way in creating large enterprise applications. Adopting a "convention over configuration" approach will let developers build a general framework that will scale to large projects.

The Saga Continues

I encourage you to dive deeper into the Maven application management tool and think about how it can really drive and enable a more modular service-based vision for your architecture. I also encourage you to look into using Maven with the other open source frameworks listed in this article. The build system is truly a reflection of your application's health in terms of maintainability and the ability to bolt on new modules in a timely manner. This article demonstrated at a conceptual level how many different open source frameworks could be glued together to create an extremely flexible layered architecture able to expose modules as services. These services can be reused in other parts of an application by merely injecting them into horizontal or upstream modules.

Acknowledgements

I would like to thank Steve Meyer and Chris Swift for invaluable editing comments and ideas.

References

- *Cargo*: http://cargo.codehaus.org/
- *Dependency Injection*: http://www.martinfowler.com/articles/injection.html
- *Hibernate*: http://www.hibernate.org/
- *Maven*: http://maven.apache.org/
- *Spring*: http://www.springframework.org/
- *Star Wars*: http://en.wikipedia.org/wiki/Star_Wars
- *XMLBeans*: http://xmlbeans.apache.org/
- *XFire*: http://xfire.codehaus.org/

Listing 1

```
<?xml version="1.0"?>
<project>
<modelVersion>4.0.0</modelVersion>
  <groupId>com.examples.droid</groupId>
  <artifactId>droid</artifactId>
  <packaging>pom</packaging>
  <version>1.0</version>
  <name>droid</name>
  <dependencyManagement>
    <dependencies>
      <dependency>
        <groupId>org.codehaus.xfire</groupId>
        <artifactId>xfire-spring</artifactId>
        <version>1.2.4</version>
      </dependency>
      <dependency>
        <groupId>com.examples.droid</groupId>
        <artifactId>droid-core</artifactId>
        <version>${project.version}</version>
      </dependency>
      …
    </dependencies>
  </dependencyManagement>
  <modules>
    <module>droid-core</module>
    <module>droid-dao-provisioning</module>
    <module>droid-ws-provisioning</module>
  </modules>
  <build>
    <plugins>
     …
    </plugins>
  </build>
</project>
```

Listing 2

```
<?xml version="1.0"?>
<project>
  <parent>
    <groupId>com.examples.droid</groupId>
    <artifactId>droid</artifactId>
    <version>1.0</version>
  </parent>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.examples.droid</groupId>
  <artifactId>droid-core</artifactId>
  <packaging>jar</packaging>
  <name>droid core</name>
  <dependencies>
    <dependency>
      <groupId>log4j</groupId>
      <artifactId>log4j</artifactId>
    </dependency>
      … … …
  </dependencies>
</project>
```

Listing 3

```
      <plugin>
          <groupId>org.apache.maven.plugins</groupId>
          <artifactId>maven-antrun-plugin</artifactId>
          <executions>
            <execution>
              <id>generate-hibernate-source</id>
              <phase>generate-sources</phase>
              <configuration>
                <tasks>

                  <!-- Enable the HibernateToolTask -->
                  <taskdef name="hibernatetool"
```

```
            classname="org.hibernate.tool.ant.HibernateToolTask"
classpathref="maven.compile.classpath"/>
     <!-- Generate XML metadata mapping files from database schema -->
                  <hibernatetool>
   …
                  </hibernatetool>

               </tasks>
             </configuration>
             <goals>
               <goal>run</goal>
             </goals>
           </execution>
         </executions>
      </plugin>
```

Listing 4

```
  <bean id="provisioningDaoImpl" class="com.examples.droid.dao.provision-
ing.ProvisioningDaoImpl">
    <property name="sessionFactory" ref="provisioningSessionFactory"/>
  </bean>

  <bean id="provisioningSessionFactory" class="org.springframework.orm.
hibernate3.LocalSessionFactoryBean">
    <property name="dataSource" ref="provisioningDataSource"/>
    <property name="mappingResources">
      <list>
       <value>com/examples/droid/dao/provisioning/Android.hbm.xml</value>
      </list>
    </property>
    <property name="hibernateProperties">
      <props>
        <prop key="hibernate.dialect">org.hibernate.dialect.HSQLDialect</
prop>
        <prop key="hibernate.hbm2ddl.auto">create</prop>
      </props>
    </property>
  </bean>
```

Listing 5

```
<execution>
  <id>generate-wsdl-types</id>
  <phase>generate-sources</phase>
  <configuration>
  <tasks>
    <!-- generate and compile types from our schema -->
    <taskdef name="xmlbean" classname="org.apache.xmlbeans.impl.tool.
XMLBean" classpathref="maven.compile.classpath"/>
     <xmlbean
        classgendir="${project.build.directory}/classes"
srcgendir="${project.build.directory}/generated-source"
        classpathref="maven.compile.classpath"
        failonerror="true"
        javasource="1.5">
        <fileset dir="src/wsdl" includes="*.wsdl"/>
     </xmlbean>
    <!-- generate our service client, interface, and impl class -->
    <!—  using XFire's web service generation task               -->
    <wsgen outputDirectory="${project.build.directory}/generated-source"
                wsdl="${basedir}/src/wsdl/${project.artifactId}.wsdl"
                package="${project.groupId}"
                generateServerStubs="true"
                binding="xmlbeans"/>
              </tasks>
              <sourceRoot>${project.build.directory}/generated-source</
sourceRoot>
  </configuration>
</execution>
```

Listing 6

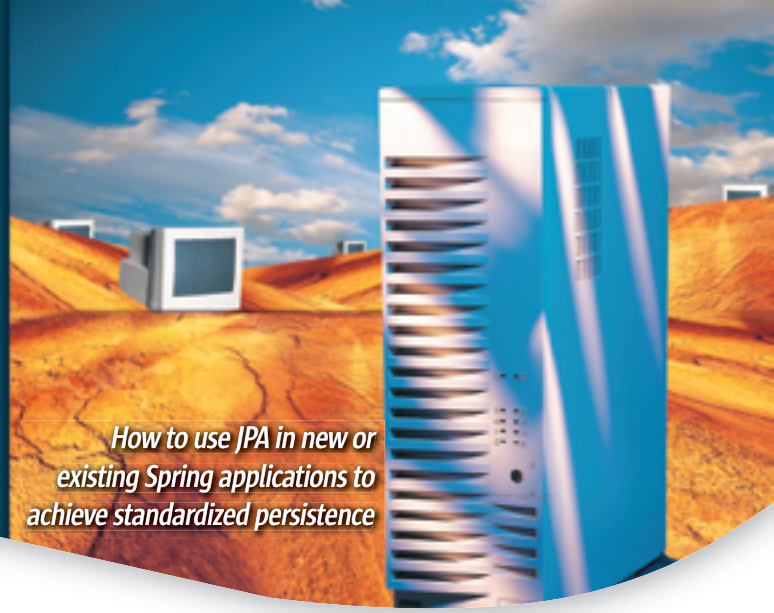
```
Public class ProvisioningServiceImpl implements ProvisioningServicePort {

  private CoreService coreService;
  private ProvisioningDao provisioningDao;

  public FindDroidResponseDocument findDroid(FindDroidDocument find-
DroidRequest) {
    FindDroidResponseDocument document = FindDroidResponseDocument.
Factory.newInstance();
    // Calling the core service to execute some functionality
    getCoreService().doSomething("do something");
    // Use the data access service and call the method findDroid()
    // to find our droid
    Android android = getProvisioningDao().findDroid(findDroidRequest.
getFindDroid().getDroidId());
    FindDroidResponse response = document.addNewFindDroidResponse();
    response.setIsError(false);
    DroidInfo droidInfo = response.addNewDroidInfo();
    // in essence this is our mapping of domain objects (Android) to
    // our message objects (DroidInfo). Never expose your domain
    // objects to the outside world.
    droidInfo.setName(android.getName());
    response.setDroidInfo(droidInfo);
    return document;
  }

  getters() and setters() for the two injected services…
}
```

Using the Java Persistence API (JPA) with Spring 2.0



How to use JPA in new or existing Spring applications to achieve standardized persistence

by Mike Keith and Rod Johnson

The EJB 3.0 Java Persistence API (JPA) was released in May 2006 as part of the Java Enterprise Edition 5 (Java EE) platform, and it has already garnered a great deal of attention and praise. What began as merely an easier-to-use successor to the much-maligned container-managed persistence (CMP) portion of the EJB component standard soon evolved into a full-blown incorporation of the existing best practices of the most prominent and popular object-relational (O-R) persistence products in use. The result is that applications now have a modern standard for lightweight enterprise Java persistence that they can use in any compliant Java EE 5 application server, or in Java Standard Edition (SE) applications.

The Spring application framework has been in existence for four years, and it has become a popular choice both in an application server context and standalone. Like JPA, Spring is a technology designed to allow applications to be built from POJOs. The Spring Framework runs within whatever runtime context an application requires, and it supports applications by providing a wide range of services. Some of these services are abstractions over existing container-level services, whereas others add value to the Java EE container. The persistence access layer, which is particularly popular with the Spring community, is nicely integrated with whatever persistence runtime is being used and facilitates a sound testable architectural approach to working with persistent objects. Spring 1.x included support for a variety of open source and commercial persistence implementations such as TopLink, Hibernate, iBATIS, and JDO, as well as the standard Java database connectivity (JDBC) API that's part of the Java runtime. Spring 2.0 was a major milestone and introduced additional integrated support for JPA. In this article we'll discuss how to use Spring and JPA together and highlight some of the benefits that this architecture can bring to an application.

Mike Keith has more than 15 years of teaching, research and practical experience in object-oriented and distributed systems, specializing in object persistence. He was the co-specification lead for EJB 3.0 (JSR 220), a member of the Java EE 5 expert group (JSR 244) and co-authored the premier JPA reference book Pro EJB 3: Java Persistence API. Mike is currently a persistence architect for Oracle and a popular speaker at numerous conferences and events around the world.

Spring as a JPA Container

The Java Persistence API was architected so it could be used both inside and outside a Java EE 5 container. When there's no container to manage JPA entity managers and transactions, the application must bear more of the management burden. When running in a JPA container the user experience is more hospitable.

One goal of the JPA specification was to make the technology pluggable. To enable this, the roles of container provider (the container or the side that has control of the runtime threads and transactions), and persistence provider (the provider or the part that implements the persistence API and manages the persistent entities) were defined, and a service provider interface (SPI) binds the two at deployment and runtime. A compliant JPA host container correctly implements this SPI from the container perspective. A compliant JPA persistence provider implements the SPI from the provider perspective. If both sides follow the rules, a compliant container should be able to run any compliant persistence provider implementation, and similarly, a provider should plug into any container.

Although Spring is neither an application server nor a Java EE 5 container, it does enhance, augment, and sometimes implement many of the services used in application servers. Spring 2.0 implements the container portion of the JPA SPI so it can be viewed as a JPA container. As such, it provides the class-loading and weaving support that JPA providers use to help manage the entities at runtime. Users benefit from an environment in which the runtime container and the JPA persistence provider are tightly integrated, but not necessarily in a Java EE 5 context. This provides many of the benefits of Java EE persistence without requiring a Java EE container.

Defining Entities

The most basic part of using JPA is to design and create the entities to be persisted. For the purposes of this article, we will use an extremely simplified library book inventory system with a single entity to illustrate the concepts concretely in Java code.

We'll create a simple **Book** entity by defining the class and annotating it with an **@Entity** annotation. The table that stores book instances will default to **BOOK**, which is exactly what we want. The primary key identifier is the **isbn** field, so we annotate that field with **@Id**. Because the **title** and **author** fields are basic mappings from the object fields to columns of the same name in the database table, we don't have to do anything to them. We want the **genre** field to map to a database column named **CATEGORY**, so we give

it a **@Column** annotation. The resulting **Book** entity class is shown below.

```
package org.bookguru;

import javax.persistence.*;
@Entity
public class Book {

    @Id private int isbn;

    private String title;

    private String author;

    @Column(name="CATEGORY")
    private Genre genre;

    // Constructors, getters and setters, etc.
}
```

Of course a real application would have many entities, but because we want to focus on the use of JPA in Spring, we won't explain how to define and map JPA entities. For more information on defining JPA entities see *Pro EJB 3: Java Persistence API*.

Using JPA Entities in Spring

The primary way to operate on entities is by using an entity manager. The **EntityManager** API is the main gateway into JPA and supports basic create/read/update/delete (CRUD) operations. It acts as both a manager of all loaded entities and a factory for queries that enable more entities to be loaded. An entity manager is analogous to an Oracle TopLink session, Hibernate session or an equivalent interface provided by many O-R mapping frameworks.

For example, to create a new persistent entity we would simply create a new Java object of the correct entity type, invoke the **persist()** method on the entity manager, and pass the new entity as a parameter. Assuming we have access to an entity manager, the code to create a new book is simple.

```
Book book = new Book(12769356, "War and Peace",
                     "Leo Tolstoy", Genre.FICTION);
entityManager.persist(book);
```

Using JPA and the entity manager in Spring is very simple. In most cases it's simply a matter of annotating a field or method of a Spring bean with **@PersistenceContext**, which causes an entity manager to be injected. Then invoke the entity manager in the context of a container transaction. Note that **@PersistenceContext** is a standard JPA annotation and not specific to Spring.

Using Spring, the transaction can be started and committed (or rolled back) at method entry and exit. All that needs to be done to achieve this is to declaratively state that the automatic transaction demarcation should happen. In Spring 2.0 the easiest way to do this is by annotating the bean class or method with the **@Transactional** annotation, although it's also possible to use XML metadata that doesn't require annotating Java code. The type of transaction that's started depends on the type of transaction manager that's configured in the Spring application context; knowledge of the underlying transaction infrastructure is completely abstracted from the Java code.

An example of a Spring bean that's transactional and uses an entity manager to perform JPA operations is the **BookInventorySystem** class shown below.

```
package org.bookguru;

import javax.persistence.*;
import org.springframework.transaction.annotation.Transactional;

@Transactional
public class BookInventorySystem {

    @PersistenceContext(unitName="BIS")
    EntityManager em;

    public void addBook(int isbn, String title,
                        String author, Genre genre) {
        Book book = new Book(isbn, title, author, genre);
        em.persist(book);
    }
}
```

This class looks fairly ordinary except that the presence of two additional annotations, **@Transactional** and **@PersistenceContext**, provides us with a great deal more functionality. The **@Transactional** annotation causes all the methods in the class to get an automatic transaction, so a transaction will be provided whenever a caller invokes the **addBook()** method. We could just as easily have annotated the method directly to get this behavior, but the likelihood of adding more methods that also need a transaction is quite high, so the class is the best place for it.

The **em** field will get injected with an instance of **EntityManager**. The entity manager injected will be configured according to the named persistence unit referred to in the **unitName** attribute of the **@PersistenceContext** annotation. Named persistence units are defined and configured in the JPA *persistence.xml* configuration file and in the Spring application context as part of the entity manager factory bean (see **Configuring the Application Context** later in this article).

Despite the sparseness of the operations (we could fill it in with more operations and queries, but it's sufficient for purposes of illustration), we have a functional system, and we can now turn to the configuration.

Configuring persistence.xml

The standard JPA configuration file is an XML file called *persistence.xml*, and it's placed in the *META-INF* directory of the jar archive or on the classpath. When using JPA in most runtime environments, this file will contain most of the runtime configuration information (except O-R mapping). However, when using JPA in Spring, this file contains only the definition of the persistence unit and sometimes a list of the entity classes (if not running in a server deployment environment). An example of a *persistence.xml* file for us is shown below.

```
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
             version="1.0">
```

Rod Johnson is the CEO of Interface21 and an authority on Java and J2EE development. He is a best-selling author, experienced consultant, and open source developer, as well as a popular conference speaker. Rod is the founder of the Spring Framework, which began from code published with Expert One-on-One J2EE Design and Development.

> The flexibility and loose coupling that Spring offers, with the standardization of JPA persistence, provides developers with the best of both worlds and gives them a platform that's easy to develop on and convenient to test"

```
    <persistence-unit name="BIS" transaction-type="RESOURCE_LOCAL">
        <class>org.bookguru.Book</class>
    </persistence-unit>
</persistence>
```

The type of transaction also depends on the deployment environment. In this example, we're running in a simple Java SE virtual machine (VM) and don't have access to a JTA transaction manager, so we set the transaction type to **RESOURCE_LOCAL**.

Configuring the Application Context

Every Spring application must eventually construct an application context — a set of bean definitions that specify the dependencies that a bean has on others. A Spring "bean" is a component in the application; it's configured by Spring and eligible to benefit from the services Spring provides. The application context determines how the beans fit together at runtime and provides the flexibility to rewire parts of an application in different ways without having to change the application Java code.

Configuring the Entity Manager Factory Bean

As part of its support for JPA, Spring 2.0 provides several JPA-related classes intended to be used as Spring beans. The most important of these is the entity manager factory bean, which makes a JPA entity manager factory available to the application. This bean has dependencies that determine the parameters of JPA execution in Spring, and although many of these settings can be defined in the JPA *persistence.xml* file, the Spring application context can provide additional flexibility and configurability. It also provides a configuration style and experience that's consistent with what Spring users are accustomed to.

Configuring the entity manager factory bean involves configuring three main dependencies: the persistence unit name, the data source, and the load time weaver. This is done as follows:

```
<bean id="entityManagerFactory"
      class="org.springframework.orm.jpa.
                        LocalContainerEntityManagerFactoryBean">
    <property name="persistenceUnitName" value="BIS"/>
    <property name="dataSource" ref="dataSource"/>
    <property name="loadTimeWeaver"
              class="org.springframework.instrument.classloading.
                        InstrumentationLoadTimeWeaver"/>
    <property name="jpaVendorAdapter" ref="vendorAdapter"/>
</bean>
```

The type of component created by this bean definition is **EntityManagerFactory**, which is the starting point for JPA usage.

The persistence unit name is just the name of the persistence unit, and the data source is defined in the usual way. Spring always uses one or more Java 2 Standard Edition (J2SE) data source definitions as the starting point for persistence configuration. A number of data source types are available, but in this case we're using a simple pooled JDBC data source defined as follows:

```
    <bean id="dataSource"
          class="org.apache.commons.dbcp.BasicDataSource">
        <property name="driverClassName"
                  value="oracle.jdbc.OracleDriver"/>
        <property name="url"
                  value="jdbc:oracle:thin:@booksvr.org:1521:
BOOKS"/>
        <property name="userName" value="scott"/>
        <property name="password" value="tiger"/>
    </bean>
```

The **loadTimeWeaver** property specifies the weaving strategy that Spring uses to implement the container provider SPI and provides the weaving capability. In this example we use the instrumentation feature introduced in the Java SE 5 VM (specified on the **jre** command line) so we specify the **InstrumentationLoadTimeWeaver** class. If we were running in a Tomcat server, we would set this to **ReflectiveLoadTimeWeaver** and use the Tomcat class loader provided by Spring. If we were running Spring inside the Oracle Containers for J2EE (OC4J) server, we would set it to **OC4JLoadTimeWeaver**, which plugs into the special classloading support in OC4J.

Configuring the Vendor Adapter

A keen observer might have noticed that we snuck an additional fourth dependency into the entity manager factory bean and wired it to a bean called **vendorAdapter**. The **jpaVendorAdapter** property is an optional property that facilitates setting vendor-specific properties that are common across providers. These properties detail:

- Whether the SQL traces should be logged
- The database platform that's being used
- Whether the schema should be generated in the database when the application starts

We added this property because the various persistence providers tend to provide a mechanism for configuring these properties, but the mechanisms differ from provider

to provider. By defining common property names in Spring, the settings can be specified regardless of the provider implementation being wired in underneath. This facilitates switching between different persistence providers by minimizing configuration changes — something that can help users determine which of the available providers exhibits the best performance or can best meet their application requirements.

The following is the definition of the **vendorAdapter** bean that we wired to the **jpaVendorAdapter** property. It defines TopLink as the vendor and gives values to the three common property settings.

```
<bean id="vendorAdapter" class="org.springframework.orm.jpa.
                                vendor.TopLinkJpaVendorAdapter">
    <property name="databasePlatform" value="${platform}"/>
    <property name="showSql" value="true"/>
    <property name="generateDdl" value="true"/>
</bean>
```

The **databasePlatform** string is understood by the persistence provider so even though the property name is common, the values may be different across vendors. We have assigned it the variable **platform** and defined it in an external application context properties file. (See *Professional Java Development with the Spring Framework* for a description of how to define and use properties files.).

Implementations such as TopLink define many more JPA settings that can be used to configure the provider in ways ranging from specifying what kind of cache to use to declaring custom classes and mapping types. These additional properties are typically defined as properties in the *persistence.xml* file.

Other Configurations

The next thing we need to do is declare the **BookInventorySystem** class as a Spring bean. The simple bean definition merely points out that Spring should proxy and manage instances of the class as they are created.

```
<bean class="org.bookguru.BookInventorySystem"/>
```

Next, we'll use the local resource-level transactions provided by the JPA entity manager, so we define the transaction manager bean and bind it to the **JpaTransactionManager** class. We then refer its entity manager factory dependency to our entity manager factory bean.

```
<bean id="transactionManager"
      class="org.springframework.orm.jpa.
JpaTransactionManager">
    <property name="entityManagerFactory"
              ref="entityManagerFactory"/>
</bean>
```

This transaction manager is designed to support transactional connections through JPA but it will also allow direct JDBC access using Spring's JDBC abstraction library.

We need to do a bit of housekeeping to indicate to Spring that it should honor and act on any **@PersistenceContext** and **@Transactional** annotations found in bean classes. This is done by adding the following two simple elements:

```
<bean class="org.springframework.orm.jpa.support.
                        PersistenceAnnotationBeanPostProcessor"/>
<tx:annotation-driven/>
```

The **tx** namespace and schema should be added to the top of the application context XML file so the namespace can be recognized.

Testing

Spring is designed to facilitate agile development practices — particularly to make testing much easier than in traditional Java EE development. The use of dependency injection and POJO programming makes unit testing much easier in general, but Spring doesn't stop there.

Spring provides a powerful integration test facility that lets code that accesses persistent data be tested without deploying to an application server or any container other than Spring. This functionality is packaged in the *spring-mock.jar* file included in the Spring distribution and provides the following services:

- Automatic transaction demarcation. Each test runs in its own transaction, which is rolled back by default. This ensures that each test can run in its own sandbox without producing side effects.
- Caching configurations. A configuration such as the O-R mappings is loaded only once, ensuring that start-up costs aren't repeated for each and every test.
- Dependency injection of test cases. Test cases can be injected just like application components, making them easier to develop and initialize.

Taken together, these services mean that tests can be written quickly and easily, and executed in seconds. The following example shows how easily we can test our book inventory system.

We extend Spring's **AbstractJpaTests** superclass, which is a subclass of JUnit **TestCase**, and then we specify the test fixture and configuration location. The test fixture will be automatically dependency-injected if we provide a setter method. We can provide any number of setter methods, but it's usually good practice to test one thing at a time. We must implement the **getConfigLocations**() method to return an array of the Spring configurations we want to load. Note that most of the configuration data is identical

> "Spring is designed to facilitate agile development practices – particularly to make testing much easier than in traditional Java EE development"

to that used in a deployed scenario, minimizing the amount of additional work needed to implement tests:

```
package org.bookguru;

import org.springframework.test.jpa.AbstractJpaTests;


public class BookInventorySystemTest extends AbstractJpaTests {

    private BookInventorySystem bookInventorySystem;

    public void setBookInventorySystem(
                          BookInventorySystem bookInventorySystem) {
        this.bookInventorySystem = bookInventorySystem;
    }

    protected String[] getConfigLocations() {
        return new String[] {"/my/path/my-spring-config.xml"};
    }
}
```

Now we can add any number of test methods. These can access data by using our data access object (DAO) fixture or the **jdbcTemplate** and **sharedEntityManager** instance variables inherited from **JpaTestCase** as follows:

```
    public void testAddBook() {
        int oldBookCount = jdbcTemplate.queryForInt(
                                    "SELECT COUNT(0) FROM BOOK");
        bookInventorySystem.addBook(12769356, "War and Peace",
                                    "Leo Tolstoy", Genre.FICTION);
        sharedEntityManager.flush();
        int newBookCount = jdbcTemplate.queryForInt(
                                    "SELECT COUNT(0) FROM BOOK");
        assertEquals("Must have added new row in BOOK table",
                     oldBookCount + 1, newBookCount);
    }
```

Here we're using JDBC queries in the same transaction to verify the correct behavior of our DAO. First we query for the number of rows in the **BOOK** table. Then we add a book, being sure to flush the current unit of work, using the **EntityManager.flush()** method. This forces the persistence provider to issue the necessary SQL update. Now we can issue another JDBC query to verify that we added an additional row. We know that our DAO doesn't merely execute without exception, but also causes the appropriate changes in our database. Those changes will be rolled back when the **testAddBook()** method has completed, so the changes won't be persisted or affect other tests.

With this approach, we can very quickly validate our O-R mappings and queries, as well as our Spring configuration. Very quick round trips mean that we can rapidly iterate as we enrich and map our domain model, identifying any problems early so that they take minimal time to rectify.

Best Practices

As we see from the example, most of the effort to develop JPA in Spring is in the initial configuration setup. Once we get the application context settled, the rest is just a matter of simple programming as we add more classes and beans. When working with multiple Spring/JPA projects it helps to have a template XML file that can be copied and modified as needed. If a specific architecture and pattern is commonly used, it may make sense to have a JPA-specific bean definition file and just file-include it in the application context file for every project. (Note that Spring allows a configuration to be split into any number of XML files.)

In the past, each persistence implementation had a different session API, and using a Spring template/DAO was helpful because it let Spring manage session-level resources and insulate the program from the vendor API. JPA is a standard API, so there's no longer a need to do this kind of shielding. In Spring 2.0 the entity managers are managed for you, so although Spring supports the same kind of DAO templates for JPA, they're no longer as necessary.

Spring DAOs also provide exception translation by mapping the various platform and database exceptions into a consistent Spring exception hierarchy. This provides the application with a normalized, unified exception-handling scheme, regardless of the particular database sitting underneath. In Spring 2.0 this facility is made available through the **@Repository** annotation, without requiring the use of DAO/template objects. Because there's currently no standard way for database exceptions to be wrapped in a JPA **PersistenceException**, this annotation will help applications process persistence exceptions and map them to specific causes. It's also particularly valuable for existing Spring applications that will migrate from proprietary data access APIs to JPA, or for mixing JPA and JDBC use in the same application.

As in any Spring application, using Spring with JPA ensures a consistent and testable programming model, whether you're deploying to a Java EE application server, a Web container such as Tomcat, or a standalone application.

Summary

In this article we've shown how to use JPA in new or existing Spring applications to achieve standardized persistence, with little effort or change in coding style. New Spring users can begin writing applications and bring their JPA experience with them. The flexibility and loose coupling that Spring offers, with the standardization of JPA persistence, provides developers with the best of both worlds and gives them a platform that's easy to develop on and convenient to test.

The JPA Reference Implementation can be downloaded from http://otn.oracle.com/jpa, and Spring 2.0 can be downloaded from www.springframework.org/download. To learn more about using JPA with Spring, see the Spring JPA documentation at http://static.springframework.org/spring/docs/2.0.x/reference/orm.html#orm-jpa.

References

- Mike Keith and Merrick Schincariol. *Pro EJB 3: Java Persistence API*. Apress, 2006.
- Rob Harrob and Jan Machacek. *Pro Spring*. Apress, 2005.
- Rod Johnson, Juergen Hoeller, Alef Arendsen, Thomas Risberg, Colin Sampaleanu. *Professional Java Development with the Spring Framework*. Wrox, 2005.
- Rod Johnson, Juergen Hoeller, Alef Arendsen, Colin Sampaleanu, Rob Harrop, Thomas Risberg, Darren Davison, Dmitriy Kopylenko, Mark Pollack, Thierry Templier, Erwin Vervaet, Portia Tung, Ben Hale, Adrian Colyer, John Lewis, Costin Leau, Rick Evans, *Spring Framework 2.02 Reference Documentation*. 2007.

JavaOne℠

OPEN POSSIBILITIES

May 8–11, 2007
The Moscone Center, San Francisco, CA
JavaOne Pavilion: May 8–10, 2007

java.sun.com/javaone

> JAVA™ TECHNOLOGY IS NOW OPEN–AND SO ARE THE POSSIBILITIES

The 2007 JavaOne℠ conference has expanded and is definitely one conference you won't want to miss. With the decision to open source Java™ technology, 2007 marks a major milestone for the Java platform. Whether your passion is scripting languages, open source, SOA, Web 2.0, mashups, or the core Java platform, this is a conference that has something for almost all technology developers.

LEARN MORE ABOUT*:

- > Scripting (JavaScript™ Programming Language, PHP, Ruby on Rails, Python, and More)
- > Open Source and Community Development
- > Integration and Service-Oriented Development
- > Web 2.0 Development
- > AJAX
- > Java Technology and the Core Java Platforms (EE/SE/ME)
- > Compatibility and Interoperability
- > Business Management

SAVE $100**
Register Today!
Please use priority code: J7PA3JDJ

* Content subject to change.
** Offer not available on-site.

Attend the JavaOne conference, and you will have many opportunities over the course of four days to network with like-minded developers; attend in-depth technical sessions; engage with your peers in Hands-on Labs and BOFs; and experience general sessions featuring speakers from Intel Corporation, Motorola, Oracle, and Sun Microsystems. Meet face-to-face with leading technology companies, and test-drive the latest tools and technologies shaping the industry.

PLATINUM COSPONSORS

intel
MOTODEV The Motorola developer network
ORACLE

GOLD COSPONSORS

bea Think liquid.
NAVTEQ
NOKIA

SILVER COSPONSORS

INTERSYSTEMS
PARASOFT We make software work.
TERRACOTTA

Sun microsystems

Minimizing the Impact of Change

Effective Business Process Management with IBM WebSphere MQSeries Workflow

by Praveen K. Chhangani

Designing and developing effective Business Process Management (BPM) solutions in conjunction with IBM's WebSphere MQSeries Workflow product involves an examination and thorough understanding of the multiple facets involved as the technical solution is being built and the business processes that serve as the foundation pillars of the execution of the business process model. In light of our fast-paced society and ever-demanding customers, rapidly interconnected markets, and wave of market globalization, it's essential for businesses to conduct their transactions efficiently.

One of the interesting facts to observe is that as technological advances and improvements occur, they change our perception of how to conduct business communications and transactions both from the business owners' point-of-view and the customer's. As I have said, "Advances in technology and the ubiquity of the Internet have realigned the way in which business functions."

The number of customers that are taking advantage of the Internet and its commerce-related benefits increases as technology and its use is simplified. Customer fingers increasingly find their way around a keyboard to type the URL of a company they want to do business with, rather than picking up the phone or reading through a brochure. This "change" is one of the reasons why businesses today need to compete harder than ever to satisfy customer demands.

Traditionally, the process of software engineering means technical engineers collaborating with business to write quality applications that help the customer conduct business more efficiently. So for example, if in the past, it was difficult for John Smith, Inc. to maintain an audit trail of all its transactions including the digital images of invoices, now with the assistance of powerful technology, technical engineers can produce quality apps that do that and more. After all, engineering software is traditionally based on a certain set of rules and conditions that, in most cases, preserves its form functionality as "fixed" or "non-adaptive." In other words, if a business is successful in doing the same kind of transactions day in and day out, an application developed for it today might still help it thrive five years from now. Where we start to see issues is when business processes have to be able to "adapt." This is where the appropriate management of how business is conducted comes into play and so the engagement of Business Process Management concepts, tools, and techniques becomes a more lucrative discussion and investment choice for businesses that want or need to better understand, track, and make their day-to-day operations more efficient.

The Effects of 'Change' on a Business Organization

Change can be a difficult and unpopular topic to address depending on an organization's culture and history. Some businesses have been conducting their daily operations the same way for the past 10, 20, even 30 years with little or no change, so when changes, especially technological changes, come about, they impact both employer and employees. And naturally every employer wants its employees to have a good, productive working environment. It's expensive for an employer to have its best workers concerned about their future because of that "new application" or "new technological method or tooling," which is more efficient, but also new. Employers may sometimes find that because of change, employee morale declines and the stress level runs high. Clearly when you start thinking of change from certain organizational perspectives, it can be a challenge to accept, initiate, maintain, and promote it within an organization.

Business Process Management as a concept provides a skeletal framework on which your applications can drive their business goals. IBM's MQWorkflow engine is an example of an application that lets a business model, simulate, and monitor its business transactions or operations via a set of interconnected relationships between people, processes, and applications.

One of the benefits of such application building is that as your business process "changes" it can be tracked better and monitored for efficiency. The degree and expense of changing the backbone of a business (i.e., its business process models) can be minimized and so this very change can be perceived as an adaptation rather than a radical change.

For example, suppose your business process involves capturing mortgage loan documents via imaging software and making a call to a service in Japan. And let's say that that service in Japan is increasingly expensive and you have a bid from another vendor that might do the same thing in Canada for half the price. You can change your business process model and replace or update only that portion of the entire business process chain. This is more effective than to, say, rewrite the whole application because one aspect of the business chain was modified.

As suggested above, another key item to understand is that "change" should be perceived as a form of "adaptation." As business owners and managers, bear in mind that a new "tool" or "process" is only good if it's used effectively to benefit the core initiatives of the business organization. It can become increasingly important to administer certain organi-

As a Certified IBM WebSphere MQWorkflow Specialist and the WebSphere Business Integration (WBI) Practice Lead for SYSCOM, Inc., **Praveen K. Chhangani** is part of the company's specialized team of WebSphere consultants whom IBM calls upon to service its most challenging customer requirements by providing training, customization, administration and configuring, architecture design, development, and deployment of distributed architectures. He has several years of experience and is well rounded as a developer, analyst, administrator and solutions architect. His extensive experience with IBM WebSphere MQSeries Workflow, WBI Modeler, Monitor and Process Choreography has proven invaluable in a full lifecycle of projects from requirement analysis, process design and automation, solution design, development to testing and mentoring for clients and customers like Principal Financial Group, MCI, State Farm Insurance and others.

praveenchhangani@yahoo.com

zation development interventions, meaning find innovative ways through which "change" is more acceptable.

Business Process Design & Architecture

Every Business Process Model is a reflection of an organization's business process, and specifically how well the business process is understood and captured for process model design. This demands an intense appreciation of the different aspects of the organization's business process and regular operating functions. Before process design it's important to capture entities such as people, process, and infrastructure into relationships that form a workflow.

As I've said before, "Workflow is a term that's typically used within the boundaries of operations and people involved in a given system. Most importantly, it revolves today around aspects of business integration methodologies by promoting process automation and business-level monitoring in real-time while supporting fluctuations in business growth."

Another aspect closer to the implementation level is the basis of your organization's architecture and management practices around it. The amount of time invested in building a robust and dynamic architecture platform will be key to the success of the business process and hence the organization. Architectures aren't what they used to be, and our business models and drivers aren't either. As our business processes and drivers grow more and more complex so does the need to improve the underlying architectures that support the models and drivers. Or, as Michael Havey says, "A good architecture uses the technique of divide and conquer to reduce a difficult problem to smaller, more manageable parts, and where possible, it solves each part not by inventing new technology but by reusing an existing approach."

Using IBM's WebSphere MQSeries Workflow

While there is a host of business process management development products available to meet your business needs, here I intend to discuss IBM's WebSphere MQSeries Workflow product in conjunction with the lifecycle of a project. The MQSeries Workflow product allows for the effective technical management of the integration of people, processes, and IT infrastructure. When used appropriately, and with strategic and researched investments, the benefits can be very rewarding. The flip side is that when it's used inefficiently or is improperly managed, it can be a difficult technical management exercise and impact your business. So I intend to elaborate on some of its intricacies.

Managing Technical Resources and Capacity Planning

It's necessary for project management to help identify and declare the vast role resources may have on the use of MQWorkflow in conjunction with the requirements of a project. In most cases MQWorkflow lies at the heart of the overall application, and hence can be a complex service to understand, build, and maintain. The proper planning of the various roles associated with an MQWorkflow project and their potential time allocations will help reduce some common project problems. Planning and preparing a resource for the right type of work is just as important as the amount of work that will be done by the resources. Technical managers should be aware of what's involved in each role and how much time is appropriate. For example, a company may have a resource path that is segregated to distinguish between an MQWorkflow developer and an MQWorkflow administrator. Another company might seek to have both roles fulfilled by one person. The

proper delegation of roles is necessary and comes from understanding the nature of the project and how MQWorkflow is being used. Many MQWorkflow developers have vast expertise. Properly identifying the experience desired for a specific project will enable higher efficiency and more productivity. Another aspect of this is maintaining resource capacities by having the necessary resource backups.

BPM Software Development, Testing, and Coordination Management

If your company project is planning on using MQSeries Workflow, it's essential to understand the nature of how development, testing, and coordination efforts are done during the course of your project's lifecycle. If your perception as a project/technical manager is that your company isn't set technically up to have a streamlined process of providing MQWorkflow services to developers for development and testing their business processes, take the time to collaborate with your peers and both project and service management, and consider investing some time in constructing a core set of services facilitated by the combination of company staff and technical products.

As we learned above, many businesses go through change to keep costs down and compete effectively and the constant change pressures management personnel to, at times, focus entirely on the business customer and his needs. While this is perhaps the most important thing to focus on, take a step back once and a while and consider whether or not your company is set up with the appropriate infrastructure to provide the services being promised to business partners and customers. Every now and then we tend to see a reflection of this, or the lack of it, in projects that didn't plan properly initially, and although they successfully built and deployed business process models into production were unable to spend adequate time looking at other aspects of building an effective workflow support competency center in the company. This is an important issue, and should be understood and effectively communicated to appropriate company sectors. Lack of this "backbone" can lead to application performance problems, support issues, service level agreement violations, and a lot of costly time spent fixing technical problems rather than benefiting from the fruit of the IT solution.

A few things to consider here are that projects should work with service teams and areas in a collaborative environment to baseline project and service expectations. Neither the project or service area alone would be able to succeed on its own, without a collaborative-style communication pattern. So ensure adequate IT infrastructure and staff are in place to promote a healthy development, testing, and execution environment.

Technical Communications and MQWorkflow

Communication should be a central part of the Workflow project's focus. Effective communication patterns between project and service analysts and direction and technical analysts will increase awareness of the technical focus, outstanding issues, potential risks, and other necessary dimensions of the workflow project that require an almost "radar-like" interception. Being able to escalate necessary items quickly and effectively, and dissolve, deprecate, or postpone unnecessary ones will come from effective communication patterns.

One thing to consider here is that communication should be involved, yet role-centric. Typically, every workflow subject matter expert (SME) has a certain level of expertise in specific segments of the product technology. Developing a knowledge plan of which resources are most effective and when is essential and can be beneficial in the long run. It used to be the case a few years ago that there were a limited number of subject matter experts who were well-rounded and able to leverage the use and benefits of the MQSeries Workflow product. This is changing and we're seeing a more diverse and experienced community providing services. Being able to capture various resource experiences and learn from them in an effort to better position our projects and service team goals will promote efficiency and heighten productivity.

Technical Observations and Recommendations

It should be the goal of every MQWorkflow developer, administrator, architect, systems designer, and general subject matter expert (SME) to ask questions of themselves and other subject matter experts regarding the use of the IBM WebSphere MQSeries Workflow product in their organizations. Specifically, how the product is being used and what can be done to improve on applications that may have already been built, as well as on-going and future development milestones. Taking the time to cover essential topics and facilitate collaborative sessions between the appropriate subject matter experts provides the kind of direction and guidance around the product that promotes a sense of its use, tailored specifically to an organization's needs respecting business process management and development initiatives. Below are some technical observations and recommendations I'd like to make that may benefit you and your organization.

1. Drive for a best-case scenario, but have a pro-active plan in case the worst happens.
2. Ensure that MQSeries Workflow is being appropriate leveraged for the benefit of the organization.
3. Ensure that effective communication is a central theme and project focus for higher productivity.
4. Ensure that the proper standardization of the workflow runtime database and its maintenance are catered to.
5. Ensure that product's features are well researched and evaluated by technical and direction teams before implementation.
6. Every business process is different. Ensure that the appropriate invo cation method of the workflow service is being used at the appropriate times (i.e., the use of the XML interface versus APIs).
7. Equip the business process models with enough error-handling logic and procedures.
8. Use properly documented function variables during development stages (i.e., _RC). When possible, try to limit the proliferation of condition expressions in the process model. Too many make the model difficult to understand and tricky to adapt and test.
9. Make use of the display functionality (as part of the product during runtime), but insert variable elements in the description field during development.
10. Devise good architecture plans. Evaluate whether your process models are being built for speed, flexibility, user friendliness, etc.
11. Consider running batch processes on different workflow configurations from user-driven workflows.

For example, batch might put a million messages in the queue and so require more processing time. In the meantime, if the UI-driven flow performs an action on the screen that also puts a message in the same queue, there may be an issue with response time. Plan for such situations ahead of time.

12. Sub-processes are expensive from the standpoint of performance. However, their use sometimes out weighs the performance impact.
13. During various testing levels, whether iterative or waterfall, it may be good to maintain detailed spreadsheet tracking, various test cases, test dates, FDL versions, etc.
14. Ensure that the various technical environments are all configured the same way when needed, especially when it comes to security.
15. Build a backup id of the almighty product administration id 'ADMIN.' This is a key item and may be impossible to recover in certain catastrophic situations.
16. Ensure that proper source code management procedures are in place.
17. Ensure that appropriate source code is being deployed. It's very important to scan FDL before deploying it for potentially unhealthy items in the workflow server. For example, items in the domain FDL that may overwrite server configuration settings.
18. Perform a Basic Workflow Unit Test (BWU) on your FDLs to understand how expensive they are from a performance standpoint.
19. When migrations are done, make sure that you have contingency plans so you can recover from any problems.
20. Ensure that a proper segmentation strategy is in place for MQWorkflow.
21. Ensure that all changes made to the process models are tracked via defects, issues, etc. It's also a good idea that a comment, version number, and author initials are inserted into the description field next to every change to make for better clarity of process tem plate names at runtime.
22. Consider using an audit trail in your process modeling. Depending on the nature of the project, an audit trail may help management make better business decisions.
23. If you make several changes to the process models and deploy several versions on a regular basis, make sure you use of the Process Template Delete utility provided by IBM.
24. Use user-friendly icons in your business process model.
25. Use the FMCINTERNALNOOPs for decision-based routines, but make sure that each use is valuable and not simply cosmetic.
26. Don't pass large chunks of data from activity to activity in your process model.
27. Have scheduled code reviews and adaptation/improvement seminars for your process models.
28. Allocate enough time for mentoring sessions for new employees, resources.
29. It's a good idea to do an overnight FDL extract on your workflow server. Take that capture and store elsewhere for disaster recovery and archival purposes.
30. Promote active participation within the workflow technical community and as a workflow analyst bring up technical issues you may face with the technical and direction teams that might be available at your company. Being able to help identify loop-holes in the technology sector can be extremely beneficial to you since it facilitates the notion of technical networking.

References

- *Organization Development & Change*. 8th edition. Cummings & Worley.
- Peter F. Sorensen Jr., Thomas C. Head, Therese Yaeger, David Cooperrider. *Global and International Organization Development*. 4th Edition.
- Praveen K. Chhangani. "Creating a Healthy, Optimized Workflow Environment." *WebSphere Developer's Journal*, Volume 3, Issue 10.
- Michel Havey. *Essential Business Process Modeling*.
- Praveen K. Chhangani. "SOA and its Impact on EAI and On-Demand." *WebSphere Developer's Journal*, Volume 4, Issue 7.
- David L Copperrider, Peter F. Sorensen, JR, Therese F. Yaeger, Diana Whiteney. *Appreciative Inquiry*.

Spring and Java EE 5

Part 1: Simplicity and power combined

by Debu Panda

The Java Platform, Enterprise Edition, or Java EE, is the most popular middleware platform for developing and deploying enterprise applications. Java EE offers developers a choice of vendors, portability, scalability, and robustness. However, it has been criticized for its complexity and its need for a lot of redundant and procedural code. In addition, lightweight frameworks such as Spring and scripting platforms such as Ruby on Rails have emerged to challenge the platform's supremacy in the middleware world.

In response, the Java Community Process has made great efforts to simplify the developer's life with Java EE's latest incarnation: Java EE 5. Innovations such as radically simplified models for Enterprise JavaBeans (EJB) and Web services have changed how enterprise applications are built using Java EE 5 technologies. Combining the robustness of the Java EE platform with lightweight frameworks such as Spring further enables developers to rapidly develop portable, maintainable enterprise applications.

In this two-part series, I will discuss how Java EE 5 simplifies enterprise application development, then uncover how you can utilize the Spring Framework to fill the gaps left by Java EE 5.

Debu Panda, lead author of the recently published EJB 3 in Action (Manning Publications), is a senior principal product manager on the Oracle Application Server development team. He maintains an active blog on enterprise Java at http://debupanda.com.

debabrata.panda@oracle.com

Simplified Programming Model with Java EE 5

Java EE 5 radically simplifies the development of enterprise applications by:

- Adopting a plain old Java object (POJO) programming standard and setting intelligent defaults for EJB components
- Eliminating the need for deployment descriptors and using Java metadata annotations for deployment settings instead
- Introducing a simplified POJO persistence model similar to Oracle TopLink and JBoss Hibernate
- Using dependency injection instead of the Java Naming and Directory Interface (JNDI) to locate resources and EJB components

Let's briefly examine these changes.

Simplified Persistence

Most developers who used the EJB 2 container-managed persistence were disappointed with its complexity and performance. As a result, POJO persistence frameworks such as Hibernate and TopLink became popular, compelling the Java Community Process to standardize a persistence API for the Java platform on a POJO persistence model.

If you've used an object-relational (O/R) mapping framework to build the persistence tier of your application, you'll notice that each framework provides three facilities:

- ***A declarative way to perform O/R mapping.*** This method, called O/R mapping metadata, lets you map an object to one or more database tables. Most O/R frameworks use XML for storing O/R mapping metadata.
- ***An API to manipulate entities (for example, to perform create, read, update, and delete – or CRUD – operations).*** The API lets you persist, retrieve, update, or remove objects. Based on the use of the API and the O/R mapping metadata, the O/R framework performs database operations on your behalf. The API shields you from writing Java Database Connectivity (JDBC) or SQL code to persist your domain objects.
- ***A query language to retrieve objects.*** This is one of the most important aspects of persistence because improper SQL statements may slow down your database. A query language also protects your applications from being cluttered with proprietary SQL, and lets you retrieve entities or objects without writing SQL SELECT statements.

The EJB 3 Java Persistence API (JPA) standardizes the use of persistence for the Java platform by providing a standard mechanism for O/R mapping, an EntityManager API to perform CRUD operations, and a way to extend the EJB query language (EJB-QL) to retrieve entities.

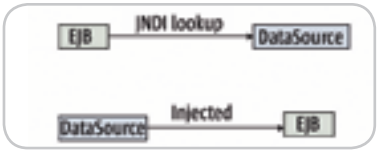


Figure 1 JNDI lookup vs. dependency injection

Introducing JPA Entities

An entity is a lightweight domain object – a plain old Java object that you want to persist in a relational database. Like any POJO, an entity may be either an abstract or a concrete class, and it can extend another POJO. You can use the @javax.persistence.Entity annotation to mark a POJO to be an entity, as shown in Listing 1.

(The code examples that follow are taken from my recently published book, *EJB 3 in Action,* published by Manning Publications.)

I've used annotations to define the mapping of entities to tables; you can also use XML. It's worth mentioning that JPA provides support for rich domain modeling capabilities such as inheritance, and polymorphism. JPA supports several inheritance mapping strategies: single table, joined subclass, and table per class. Unlike EJB 2 container-managed persistence (CMP), JPA is simple and supports automatic generation of primary keys.

Now that you've seen an entity, let's examine how you can manipulate entities by using the EntityManager API.

The EntityManager API

The javax.persistence.EntityManager manages entity lifecycles and exposes several methods to perform CRUD operations on entities. JPA supports two types of EntityManager: container-managed and application-managed. The application-managed EntityManager is really useful when using JPA outside a container. Let's look at an example of using a container-managed EntityManager to manage an entity.

You can use the persist() method to save an instance of entity. For example, if you want to persist an instance of Bid, use the following code:

```
@PersistenceContext(unitName="actionBazaar")
private EntityManager em;
```

2007 VIRTUALIZATION CONFERENCE + EXPO

www.virtualizationconference.com

Register Now!
EARLY-BIRD SAVINGS!
$200

ROOSEVELT HOTEL
NEWYORK CITY
June 25-27, 2007

Virtualization: Solutions for the Enterprise.

Delivering The #1 i-Technology Educational and Networking Event of the Year!

As today's CIOs and IT managers rise to the challenge of using their enterprise computing infrastructure as cost-effectively as possible while remaining responsive in supporting new business initiatives and flexible in adapting to organizational changes, Virtualization has become more and more important.

A fundamental technological innovation that allows skilled IT managers to deploy creative solutions to such business challenges, Virtualization is a methodology whose time has come. The fast-emerging age of Grid Computing is enabling the virtualization of distributed computing, of IT resources such as storage, bandwidth, and CPU cycles.

But Virtualization can also apply to a range of system layers, including hardware-level virtualization, operating system level virtualization, and high-level language virtual machines.

Register Online!
www.VirtualizationConference.com

THE FIRST MAJOR VIRTUALIZATION EVENT IN THE WORLD!

HURRY, FOR EXHIBITOR AND SPONSORSHIP OPPORTUNITIES
CALL 201-802-3020

— BROUGHT TO YOU BY —
ENTERPRISE OpenSource MAGAZINE
SOAWORLD MAGAZINE

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

```
...
Bid bid = new Bid();
bid.setItem(item);
bid.setBidder(bidder);
bid.setBidPrice(price);
em.persist(bid);
```

Now that you have a sense of how easy it is to use the persistence feature of EJB 3, we'll examine how EJB 3 simplifies the development of business components.

Simplified EJB 3 Components

EJB 2 was one of the primary technologies responsible for the complexities that have plagued enterprise Java development. Some detractors ridiculed it as a "fat elephant" for its heavyweight nature. It required a lot of redundant code, even to build a simple "HelloWorld" EJB.

EJB 3 simplifies development by adopting the POJO programming model, and simplifies usage of EJB and resources by using dependency injection. It also depends heavily on intelligent defaults and makes the deployment descriptor optional.

Listing 2 provides an example of a simple stateless EJB 3 session bean with a remote interface. In this example, PlaceBidBean is a simple POJO class that implements a regular Java interface – or a plain old Java interface (POJI). The @javax.ejb.Remote converts the POJI to a remote interface and @javax.ejb.Stateless converts the POJO to a stateless EJB.

You can use @Stateful and @MessageDriven annotations to define stateful and message-driven beans, respectively.

Dependency Injection

With Java EE 5, invoking an EJB component is a snap with dependency injection. Dependency injection support in Java EE 5 was influenced by inversion of control (IoC) containers such as Spring.

Dependency injection is essentially the opposite of the JNDI: the container is responsible for looking up and instantiating an instance of an EJB or a resource. Figure 1 depicts how dependency injection compares with JNDI.

Spring Class	Purpose
JpaTemplate	Simplifies JPA access code
JpaDaoSupport	Super class for Spring DAO
LocalEntityManagerFactoryBean	Factory that creates local entity manager when JPA inside Spring container
JpaDialect	Used with a persistence provider outside Java EE

Table 1 Spring classes supplied for JPA

For example, in Figure 1 you can invoke the EJB from a managed component as follows:

```
@EJB
private PlaceBid placeBid;
..
    Long bidId = placeBid.addBid(bid);
..
```

Although Java EE 5 simplifies usage resources and services by using dependency injection, it is supported only on managed classes such as Java servlets, listeners, and EJBs, and not regular Java classes. Later I'll discuss how you can use Spring's powerful dependency injection features to fill that gap.

Note that Listing 2 uses JMS Queue to send messages. The @Resource annotation injects an instance of JMS connection factory and destination, respectively, without having to do a complex JNDI lookup. Usage of JMS is fairly complex, but Spring simplifies it with its JmsTemplate, as we will see in the second part of this article.

Simplified Web Service

Development of Web services was incredibly complex with the Java API for XML-based RPC (JAX-RPC), which was part of earlier versions of J2EE. The Java API for XML Web services (JAX-WS), part of the Java EE 5 platform, simplifies the development and invocation of Web services by using annotations. You can easily expose any POJO or stateless EJB as a Web service by using the @WebService annotation. For example, you can expose the PlaceBid EJB as shown in Listing 3.

If you have used Web services in J2EE 1.4, you'll realize how simple it is to develop Web services in EJB 3!

The invocation of Web services was also a complex task with J2EE 1.4. Java EE 5 simplifies it by using @WebServiceRef annotations, as shown here:

```
@WebServiceRef(wsdlLocation="http:
//localhost:8888/PlaceBidService/
PlaceBidService?WSDL")
  private static PlaceBidService placeBid
    Service;

        actionbazaarplacebidservice.
PlaceBidBean placeBid = placeBidService.
    getPlaceBidBeanPort();
            System.out.println("Bid
Successful, BidId Received is:" +placeBid.
addBid("idiot",  Long.valueOf(1),  2000005.50
));
```

Again, however, @WebServiceRef injection is only supported with managed classes and cannot be used from regular Java objects.

In the first half of the article, I discussed how Java EE 5 simplifies development of enterprise Java applications. Now I'll explore how you can use Spring to address limitations in Java EE.

Integrating the Power of Java EE 5 and Spring

The Spring Framework provides a lightweight container with functionality that simplifies the development of applications. Although many view Spring as an alternative to Java EE, most customers use it as a framework and deploy Spring-based applications into a Java EE container such as Oracle Containers for J2EE (OC4J).

Spring simplifies resource access by using dependency injection, and simplifies database access by using a template-based approach. In this section, I'll take a look at these capabilities.

JPA and Spring

Spring has wide support for data access that includes popular object-relational mapping (ORM) frameworks, including Hibernate and TopLink. Spring takes a data access object (DAO) approach to coding, allowing developers to use these ORM options and switch between them easily. While JdbcTemplate makes database access using JDBC simpler, Spring 2.0 has built-in integration with JPA, and it ships TopLink Essentials (the reference implementation of JPA derived from Oracle TopLink in Sun's GlassFish project) as the default JPA provider.

While Spring supports container-managed EntityManager as a lightweight container, it simplifies usage of JPA with JpaTemplate. Table 1 shows several Java interfaces for using JPA in Spring.

Spring enables you to access and manipulate entities either using JPA directly or using JpaTemplate. I will now demonstrate how to use JPA from applications using JpaTemplate. Listing 4 shows a DAO implementation class using JpaTemplate.

You can use the JpaTemplate methods to access entities. Spring really simplifies some of the repetitive usage of EJB 3 JPA. For example, if you want to use a dynamic query to retrieve all bids for an item, you can use the JpaTemplate to produce this:

```
List bids = getJpaTemplate().find(
                "SELECT b FROM Bid b WHERE
b.item = ?1",item);
```

> *Businesses that ignore the potential of SOA will find themselves outpaced by rivals who improve their agility and transform themselves into new kinds of enterprises*
>
> — Yafim Natis, Gartner Analyst

3-DAY EVENT!

SOAWorld 2007

Plus

Enterprise OpenSource Conference & Expo 2007

TOPICS INCLUDE:

SOA Web Services

- AJAX and SOA
- Web 2.0
- Universal SOA
- Protecting Web Services
- Troubleshooting SOA
- Governance
- Open-Source SOA
- XBRL
- Service Virtualization

Open Source

- Open Source Business Models
- Open Source ESB
- OpenAjax Alliance
- SaaS and Open Source
- Spring, Hibernate and Eclipse
- Seam
- Open Source Penetration
- Monetizing Open Source
- Open Source Databases
- AMQP
- Open Source Middleware

June 25-27, 2007
Roosevelt Hotel / New York City

Register Online! www.SOAWorld2007.com

SOAEOSCONFERENCE.SYS-CON.COM
REGISTER ONLINE TODAY
SAVE $200!
(HURRY FOR EARLY-BIRD DISCOUNT)

11th International SOAWORLD 2007 CONFERENCE & EXPO

2nd Annual ENTERPRISE OPENSOURCE 2007 CONFERENCE+EXPO

2007 is to many industry insiders shaping up to be a major inflection point in software development and deployment, with SOA, Web Services, Open Source, and AJAX all converging as cross-platform and cross-browser apps become the rule rather than the exception.

Accordingly the 11th International SOA Web Services Edge 2007 again seeks to offer comprehensive coverage and actionable insights to the developers, architects, IT managers, CXOs, analysts, VCs, and journalists who'll be assembling as delegates and VIP guests in The Roosevelt Hotel in downtown Manhattan, June 25-26, 2007

Co-located with the 2nd Annual Enterprise Open Source Conference & Expo, the event will deliver the #1 i-technology educational and networking opportunity of the year. These two conference programs between them will present a comprehensive view of all the development and management aspects of integrating a SOA strategy and an Open Source philosophy into your enterprise. Our organizing principle is that delegates will go away from the intense two-day program replete with why-to and how-to knowledge delivered first-hand by industry experts.

Visit soaeosconference.sys-con.com for the most up-to-the-minute information including... Keynotes, Sessions, Speakers, Sponsors, Exhibitors, Schedule, etc.

BROUGHT TO YOU BY:

SOA Web Services Journal
focuses on the business and technology of Service-Oriented Architectures and Web Services. It targets enterprise application development and management, in all its aspects.

Enerprise Open Source Magazine
EOS is the world's leading publication showcasing every aspect of profitable Open Source solutions in business and consumer contexts.

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

Exhibit and Sponsorship Info:
Call 201-802-3020 or email events@sys-con.com

The equivalent code, if you use the EntityManager API, will look like this:

```
List bids = em.createQuery(
                    "SELECT b FROM Bid b
WHERE b.item = ?1")
                  .setParameter(1,item)
                  .getResultList();
```

The real power of Spring comes from how it configures services via dependency injection. To use JPA with Spring, you need the configuration shown in Listing 5.

JPA is not the only cool thing about EJB 3; there is more meat in it, and there are more integration points between Spring and EJB 3 that we will discover in the next part of this series.

Conclusion

In this article, you learned how Java EE 5 simplifies the development of enterprise-level applications. You saw examples of JPA entities, EJB 3 session beans, and a simple JAX-WS Web service. All these components are POJOs and make heavy use of annotations. JPA greatly simplifies the building of persistence applications.

The Spring Framework 2.0 has not only integrated with JPA, but also greatly simplifies usage of JPA by using JpaTemplate. In the next part of the series we will discover how the Spring Framework integrates with other components such as an EJB, Java Message Service (JMS), and transaction manager. Stay tuned!

Listing 1: A sample entity

```
@Entity
@Table(name="BIDS")
public class Bid implements Serializable {
    @Column(name="BID_DATE")
    private Date bidDate;

    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    @Column(name="BID_ID")
      private Long bidId;
    @Column(name="BID_PRICE")
     private Double bidPrice;
    @ManyToOne
    @JoinColumn(name="BID_ITEM_ID",
                          referencedColumnName="ITEM_ID")

     private Item item;
    private BidStatus bidStatus;
    …     public Bid() {
    }



    public Long getBidId() {
        return bidId;
    }

....}
```

Listing 2: A sample EJB 3 session bean

```
@Remote
public interface PlaceBid {
    Bid addBid(Bid bid);
}


@Stateless
public class PlaceBidBean implements PlaceBid {
@Resource(name = "jms/BidQueue")
    private static Destination destination;

@Resource(name = «jms/QueueConnectionFactory»)
    private static ConnectionFactory;

public PlaceBidBean() {
    }


public Bid addBid(Bid bid) {

}


 }
```

Listing 3: Simple EJB 3 Web service

```
@WebService(serviceName="PlaceBidService",
          targetNamespace = "urn:ActionBazaarPlaceBidService")
@SOAPBinding(style = SOAPBinding.Style.DOCUMENT)
@Stateless
public class PlaceBidBean {


    @WebMethod

    @WebResult(name = "bidNumber")
    public Long addBid(String userId,

      Long itemId,
      Double bidPrice) throws BidException {
            ..
}
```

Listing 4: DAO class using JpaTemplate

```
public class BidSpringEAO extends BasicSpringEAO implements BidEAO {



    public Bid addBid(Long itemId, String bidderId, double bidPrice) {
      Bid bid = new Bid();
      Item item =  (Item)this.getJpaTemplate().find(Item.class,itemId);
      bid.setItem(item);
      bid.setBidPrice(bidPrice);
      bid.setBidStatus(BidStatus.NEW);
      Bidder bidder =  (Bidder)getJpaTemplate().find(Bidder.
class,bidderId);
      …
      bid.setBidder(bidder);
      this.getJpaTemplate().persist(bid);

      return bid;
    }



   }
```

Listing 5: Spring configuration for using JPA

```
<bean id="entityManager"
  class="org.springframework.jndi.JndiObjectFactoryBean">
    <property name="jndiName">
     <value>java:comp/env/actionBazaar</value>
 </property>
     <property name="resourceRef">
     <value>true</value>
   </property>
</bean>

 <bean id="bidService" class="actionbazaar.buslogic.BidServiceBean">
    <property name="bidEAO">
      <ref bean="bidEAO"/>
    </property>
    <property name="itemEAO">
      <ref bean="itemEAO"/>
    </property>
  </bean>


 <bean id="bidEAO"
    class="actionbazaar.persistence.eao.BidSpringEAO"
    autowire="byType">
     <property name="entityManager" ref="entityManager"/>
</bean>
```

JAVA DEVELOPER'S JOURNAL Advertiser Index

Advertiser	URL	Phone	Page
Altova	www.altova.com	978-816-1600	Cover II
Extentech	http://extentech.com/itsg	415-759-5292	9
Infragistics	www.infragistics.com/jsf	800-231-8588	4
InterSystems	www.intersystems.com/ja1p		Cover IV
Java Developer's Journal	www.jdj.sys-con.com	888-303-5282	33
JavaOne	www.java.sun.com/javaone		23
Jinfonet Software	www.jinfonet.com/live	240-477-1000	7
Northwoods Software Corp.	www.nwoods.com	800-434-9820	27
Oracle	www.oracle.com/middleware	800-ORACLE-1	11
SOA and EOS 2007 Conference & Expo	www.soaworld2007.com	201-802-3020	31
Software FX	www.softwarefx.com	561-999-8888	Cover III
Virtualization Conference & Expo 2007	www.virtualizationconference.com	201-802-3020	29
VX30	www.vx30.com	866-661-5699	25

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.

JSR Bookmarks at the 2007 JavaOne Conference

Onno Kluyt

Interested in getting the latest on Java technology standards at the 2007 JavaOne Conference? A great opportunity is awaiting you because the show has a lot to offer this year too. There are over 60 events – technical sessions (TS), Birds-of-a-Feather meetings (BOF), and Hands On Labs – based on Java specifications developed or in development through the JCP. Here are a few of them presented by none other than the Spec Leads.

JSR 296: The Swing Application Framework is the topic of **TS-3942**. Joshua Marinacci and Hans Muller, both of Sun Microsystems, will present the advantages developers will be able to draw from having a standard architecture for desktop applications included in the Java platform. At the end of its development road, this specification will define a simple application framework for Swing applications. Another Swing-related session is Swing in a Multithreaded World, **TS 3565,** from which you will learn about best practices and common patterns in writing multithreaded desktop applications.

Another JSR-based session is **TS-4225, What's New in the Java Portlet Specification 2.0 (JSR 286)?** IBM's Stefan Hepper and Sun's Wesley Budzjwojski will present the functionality that will be added to the new portlet specifications. If you want to learn about the major new concepts, such as coordination between portlets, serving resources through the portlet, supporting AJAX use cases with portlets, portlet filters, and validation-based caching, plan to attend. The session will also cover other small but important changes and the alignment with the new version of Web Services for Remote Portlets (WSRP 2.0) and Web frameworks such as JavaServer Faces technology.

If you are into developing mobile applications that integrate Web services or are planning to start, bookmark **TS-5188 Web Services to Go: Mobile Access to Web Services with JSRs 279 and 280.** Spec Leads from Nokia and BenQ will present how JSRs 280 (XML API for Java ME) and 279 (Service Connection API for Java ME) will help with interacting with Web services from mobile devices by adding platform support for XML and Web services, creating a new mobile Web services ecosystem. JSR 280 provides a general-purpose API for XML processing, extending JSR 172 and adding Streaming API for XML (StAX) and Document Object Model (DOM) parsing. JSR 279 is building on JSR 280's XML API and provides a framework for straightforward access to networked services, incorporating support for service discovery, authentication, and identity.

A project recently submitted to the JCP, **JSR 311**, will be showcased in **TS-6411, The Java API for RESTful Web Services**. The spec leads will talk about the goal of the Java API for RESTful Web services, which is to provide a high-level declarative programming model for such services that is easy to use and encourages development. Services built with the API, the spec leads will show, will be deployable by use of a variety of Web container technologies and will benefit from built-in support for a variety of HTTP usage patterns and conventions.

If you're looking for a brief REST primer and an update on the progress of the JSR 311 to date, this is the session for you. The Spec Leads will outline the current API design and highlight issues currently under discussion by the expert group. Live code demonstrations will illustrate the API discussion.

Michal Cierniak, of Google, and Alex Buckley, of Sun Microsystems, partner to bring us a BOF session about the **Modularity in the Next-Generation Java Platform, Standard Edition (Java SE): JSR 277 and JSR 294**. The two Java Specification Requests (JSRs) are targeted to be delivered as a component of the Java Platform, Standard Edition (Java SE) 7.0. JSR 294 sets out to define the modules for development, and JSR 277 to define the modules for deployment. The specifications set out to address many issues including those associated with Java Archives (JARs), including the lack of version control, the difficulties in distributing multiple JARs for deployment, the classpath hell, JAR hell, and extension hell, and so on, which have been well known to many developers on the Java platform for years.

Gavin King of Red Hat Middleware, Spec Lead, and Bob Lee of Google, Expert Group member, will give a Web Beans update in **TS-4089**. JSR 299, Web Beans, aims to unify the JavaServer Faces technology-based managed bean component model with the Enterprise JavaBeans (EJB) component model, resulting in a significantly simplified programming model for Web-based applications. The two plan to provide attendees with background on the Web Beans effort, give an update on Expert Group membership, and outline the purpose and scope of the Web Beans specification, the Web Beans programming model, the impact on other JSRs: EJB 3 architecture; JavaServer Faces platform; Java EE. If you are interested in the current status of JSR 299 and the open issues with which the Spec Lead and Expert Group members are confronting themselves, plan to attend this session.

Developers who have tracked **Java Business Integration** aka JBI will want to bookmark **BOF-8872** driven by Peter Walker of Sun and Mark Little of Red Hat Middleware. The session brings together members of the Java Business Integration (JBI) 2.0 Expert Group, members of the Open ESB community, and others interested in finding out more about JBI 2.0 and its progress to date to discuss priorities and directions for work within the context of the JSR. A related technical session is **TS-8216, Why do I need JBI when we have BPEL?**, presented by Sun's Peter Walker and Andreas Egloff. Expect the session to tackle one of the most popular questions posed to the JBI team at last year's JavaOne Conference: Does JSR 208, Java Business Integration (JBI), compete or overlap with the Business Process Execution Language (BPEL)? The session will also address some common questions and misconceptions about how to best utilize JBI and how different users can get started on benefiting from JBI.

In December 2006 the Spec Leads and Expert Group of JSR 248, Mobile Services Architecture, finalized the standard. At this year's JavaOne, Kay Glahn of Vodafone and Miklos Kelen of Nokia will be presenting the session **JSR 248: Taking Java Platform, Micro Edition (Java ME) to the Next Level**, showcasing experiences with the completed JSR 248, and covering the specification already published and the status of its industry acceptance. If you want to get a JSR 248 post release update and an overview of the latest 248-compliant mobile devices, then you should plan to attend this session.

If you're looking for a hands-on lab on real-time Java programming, you've got your chance with **LAB-7250, The Real-Time Java Programming Challenge: How to Build Real-Time Solutions for Real-World Devices**. Greg Bollella and David Holmes, of Sun Microsystems, will demonstrate that Java is ready for real time and will challenge participants to see for themselves by building their own real-time Java application.

The lab is based on JSR 1, The Real-Time Specification for Java (JSR 01), which provides several key application interfaces that enable developers to create programs with predictable timing and deterministic program execution.

This is just a selection of the JSR-based sessions JavaOne has in store for you this year; the list goes on and on. I encourage you to go to the JavaOne Conference page at http://java.sun.com/javaone/sf/index.jsp, click on Content Catalog, and start bookmarking your portfolio of preferred sessions.

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

onno@jcp.org