DEVELOPING AN ECLIPSE BIRT REPORT ITEM EXTENSION PAGE 52

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

JDJ

JDJ.SYS-CON.COM VOL.11 ISSUE:10

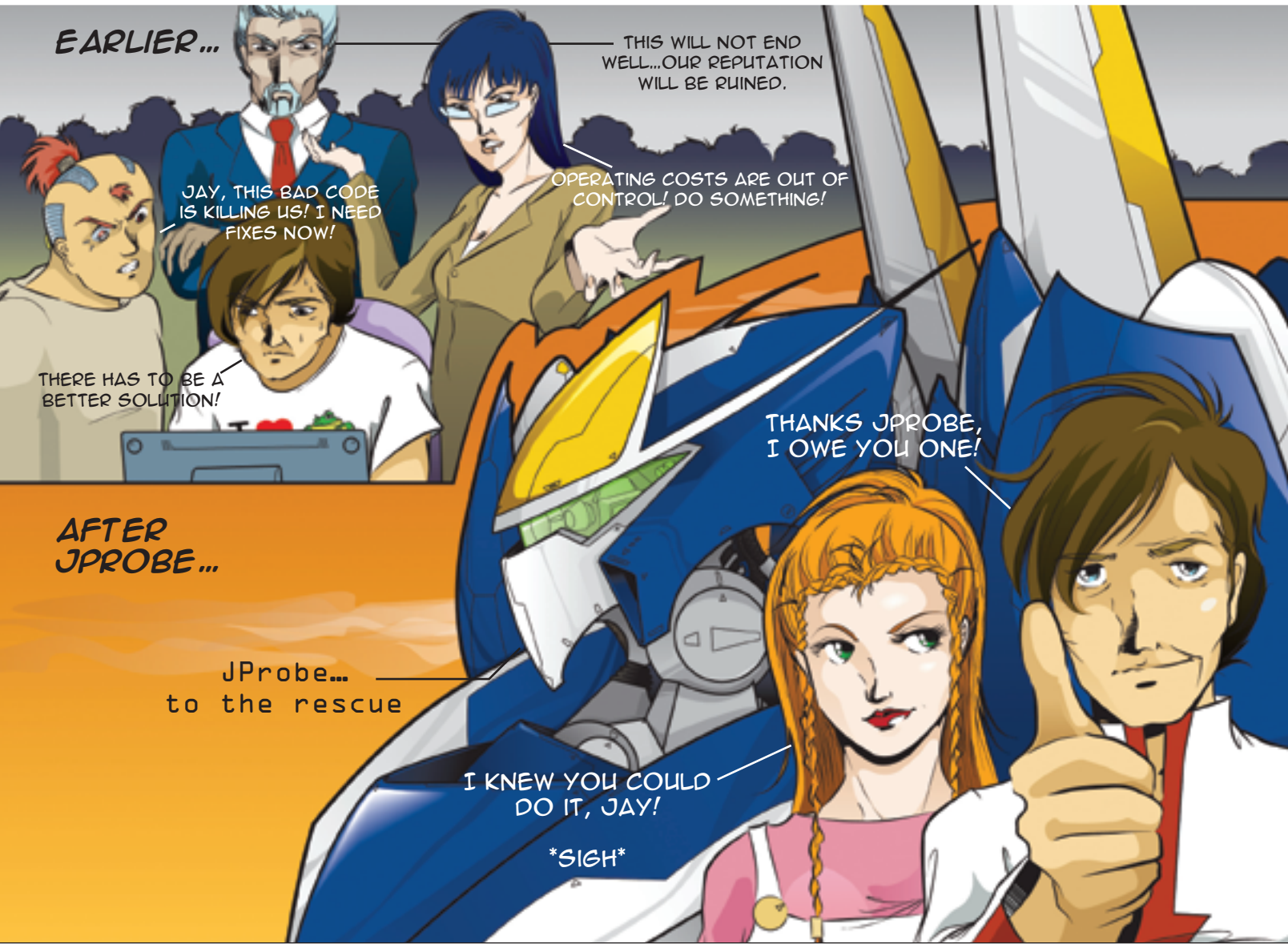
AJAX THE EASY WAY

With Java and DWR

RETAILERS PLEASE DISPLAY
UNTIL DECEMBER 31, 2006
$5.99US $6.99CAN
11
0 09281 01751 6

PLUS...

- Portable Persistence Using the EJB 3.0 Java Persistence API
- Inheritance Hierarchies in JPA
- Building a Simple VocabBuilder Application



* Free t-shirt offer

In the Battle Against Bad Java Code…

JProbe® is on your side.

Jay, our Development Manager, was in trouble. He had the Application Business Owner, Production IT Manager and even the CIO on his back about bad code finding its way into production. Operation costs were skyrocketing, productivity was down and customer loyalty was at risk.

Then Jay discovered JProbe® from Quest Software. With JProbe, Jay's team can proactively zero in on the code that affects performance – before the code goes to production. Now Jay is a hero and delivers quality, high-performing code on time – every time.

Join the battle against bad Java code in production. Download a trial of JProbe at www.quest.com/hero — and get a free JProbe t-shirt!*

QUEST SOFTWARE®

JAVA DEVELOPER'S JOURNAL

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

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

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

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

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

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

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

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

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

AJAX – What's In It for Java

Jeremy Geelan

When the fast-paced, three-day program of AJAXWorld Conference & Expo in the Santa Clara Convention Center finally ended earlier this month, with over 90 technical sessions and presentations from leading AJAX vendors like Laszlo Systems, JackBe, and Backbase as well as from established software giants like IBM, TIBCO, and Adobe, the overwhelming impression delegates, speakers, and sponsors alike were left with was of having been in attendance at something special, something unusual, something potent.

"The AJAX Moment," as I have been calling it for several months now, was almost palpable in California for those three days. So it was no surprise that literally dozens and dozens of companies chose to make announcements timed to capitalize on the vast attention that the Conference attracted in the world's media, nor was it surprising that the OpenAjax Alliance should have timed the most important membership meeting of its history – the meeting at which it elected its first-ever Steering Committee – to coincide with AJAXWorld 2006.

The fact that the OpenAjax meeting was hosted by Sun, however, might initially surprise a few readers – especially since Sun didn't win any of the seven positions on the committee. But there is plenty of interest in AJAX at Sun, even if it came a tad late in the game – so late that a Distinguished Sun Engineer like John Crupi was able to be lured away to become CTO of nimble startup JackBe, and so late that he was able in turn to entice Deepak Alur and Dan Malks (also both Java experts from Sun) to join him.

No company on earth had more speakers on the inaugural AJAXWorld speaker faculty than Sun, including Greg Murray – Servlet 2.5 Specification Lead, now the AJAX architect for Sun. Greg spent several weeks preparing for that OpenAJAX Alliance membership meeting at Sun's Santa Clara headquarters.

Other Sun speakers on the AJAXWorld 2006 faculty included Inderjeet Singh, who gave a session on "Java EE 5 BluePrints for AJAX-Enabled Web 2.0 Applications" and is a senior staff engineer with Sun Microsystems where he is the architect for the Java BluePrints program; David Van Couvering ("Embedding a Database in the Browser: Enabling Offline AJAX"), the original architect for the Sybase J2EE application server and for the first release of the clustered Sun Java Application Server Enterprise Edition; Craig McClanahan ("Encapsulating AJAX Functionality in JavaServer Faces Components"), original author of Apache Struts, part of the expert group that defined the servlet 2.2, 2.3 and JSP 1.1, 1.2 specifications, and the architect of Tomcat's servlet container Catalina; and Francois Orsini ("Apache Derby - A Local AJAX Data Store"), a senior staff engineer working in the database technology group at Sun with 18 years' experience in databases and infrastructure development.

While stopping short of trying to fool people into believing that the J in AJAX stands not for JavaScript but for Java, Sun is absolutely committed by the look of things to making sure that developers and IT architects are aware of how AJAX is agnostic as to which server-side language is being used, be it Java, ColdFusion, PHP, .NET, Perl or whatever.

Sun wants Java to be a part of "The AJAX Moment" – and a big part, at that.

AJAXWORLD™ CONFERENCE & EXPO

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

jeremy@sys-con.com

Altova® DatabaseSpy™ 2007 – The unique multi-database query and design tool.

ALTOVA®
databasespy™
2007

Evade constraining query tools

Move up to DatabaseSpy 2007, and manage all your databases from one elegant interface.

BRAND NEW DATA MANAGEMENT TOOL!

- Connects to all major databases
- Project Manager organizes connections, and Database Browser clearly presents tables, views, and data
- SQL Editor facilitates query writing with code completion, syntax coloring, drag & drop construction, and more
- Design Editor enables graphical design and visualization of database structures

Altova DatabaseSpy 2007 is the unique multi-database query and design tool from the creators of XMLSpy. A real steal, DatabaseSpy liberates data management, delivering advanced, coherent capabilities at a fraction of the cost of single-database solutions. Write accurate queries with confidence and get clear, easily negotiable results. Create and edit database structures visually by dropping tables on the design pane and dragging relationship links between them. Even duplicate existing structures in diverse database types. Do data differently! **Download DatabaseSpy™ 2007 today: www.altova.com**

Join Altova at
Oracle OpenWorld,
San Francisco
Booth #1730

ALTOVA®

OCTOBER 2006 VOLUME:11 ISSUE:10

JDJ contents

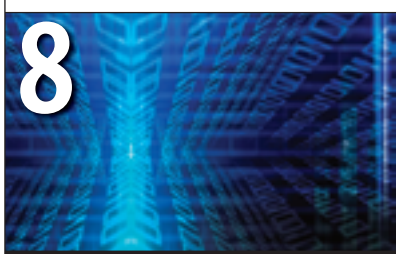
JDJ Cover Story

AJAX THE EASY WAY

With Java and DWR

by Jon Hoffman 30

Features



Portable Persistence Using the EJB 3.0 Java Persistence API
by Mike Keith and Merrick Schincariol

Inheritance Hierarchies in JPA
by Raghu R. Kodali and Jonathan Wetherbee

Developing an Eclipse BIRT Report Item Extension
by Jason Weathersby, Iana Chatalbasheva, and Tom Bondur

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

Industry News

SAP NetWeaver Achieves Java EE 5 Compatibility

(*Walldorf, Germany*) –SAP AG has announced it has achieved Java Platform, Enterprise Edition (Java EE) 5 compatibility. SAP NetWeaver has been providing complete support for Java technology since 2003. Achieving compatibility means SAP customers and partners can develop robust Java applications on the SAP NetWeaver platform using the latest mature technology standards – simplifying and accelerating application development projects. A preview of SAP's Java EE 5 implementation is available for download on SAP Developer Network at http://www.sdn.sap.com/.

Java EE 5 represents a significant step forward to simplify the Enterprise Java programming model. The design goals of Java EE were to streamline features and convenience to the platform, thereby improving performance and reducing development time. For SAP customers and partners developing applications on the SAP NetWeaver platform, compatibility of SAP NetWeaver Application Server with Java EE 5 will enable them to speed time-to-market with enterprise service-oriented architecture–based applications and reduce overall total cost of ownership (TCO). It also provides out-of-the-box connectivity to all SAP systems.

IBM Helps Businesses Increase Productivity with New Web Content Management Software

(*Armonk, NY*) – IBM has announced the availability of IBM Workplace Web Content Management 6.0, offering businesses simple customization tools that accelerate content creation and provide tight integration with IBM WebSphere Portal.

The new solution delivers easy-to-use authoring tools, making it simple for non-technical users to create and publish customized, up-to-the minute Web content. New ease-of-use features include menu-based personalization tools that allow users to customize forms based on a role or function. In addition, a choice of rich text editors and wiki-like editing features help users edit objects and data on-the-fly within the context of a Web site or portal. www.ibm.com

McObject Releases Perst Lite

(*Issaquah, WA*) – McObject has released Perst Lite, a micro-footprint version of the Perst open source, object-oriented embedded database. Perst Lite targets embedded systems and intelligent devices developed on the Java 2 Platform, Micro Edition (J2ME), bringing object-oriented database services to the fast-growing embedded Java development community.

Perst Lite features include: B-tree, Patricia Trie, Bit index, T-Tree and R-Tree indexes as well as List, Relation, and Set collections, all protected by transactions supporting the ACID properties (Atomicity, Consistency, Isolation and Durability). It also offers multithreaded access, data encryption and asynchronous replication.

Under Perst's dual license, users can modify Perst and Perst Lite database source code and use it freely in other open source applications (software for which source code is made available) under the GNU General Public License (GPL). McObject's commercial license is required if source code of the Perst- or Perst Lite-based applications is to be withheld.

More information, as well as Perst Lite and Perst database software for Java, is available from www.mcobject.com/perst.

Real-Time Java Technology Monitors Traffic, Patients and Products

aJile Systems Inc., a company founded by the developers of a direct execution Java technology microprocessor, announced three deployments of its just-released Java M2M (machine-to-machine) edge controller at the CTIA conference this week. aJile has been selected by InfoTek Wireless (www.infotekwireless.com) to provide controllers for a 100% Java technology remote traffic monitoring system; Rosonix (www.rosonix.com) has opted for aJile technology for its remote bio-monitoring systems; and AssetPulse (www.assetpulse.com) has chosen aJile to deploy its 100% Java technology RFID-enabled asset tracking solution.

By offering a Java technology edge controller, aJile enables developers to seamlessly integrate their core enterprise applications with remote edge applications using a single development language. And by using a single language – Java – that enjoys wide acceptance, a substantial ecosystem of third-party developers, a large pool of Web-based applications, industry-standard APIs and protocols, and programming environments on every PC platform, developers can reduce development time and maintenance costs, while improving reliability, portability, reusability and system security. www.agile.com

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*
Robyn Forma *robyn@sys-con.com*

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

Associate Sales Manager:
Kerry Mealia *kerry@sys-con.com*
Lauren Orsi *lauren@sys-con.com*

Editorial

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

Associate Editor:
Lauren Genovesi *laureng@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*
Mandy Eckman *mandy@sys-con.com*

Web Services

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

Web Designers:
Stephen Kilmurray *stephen@sys-con.com*
Paula Zagari *paula@sys-con.com*

Accounting

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

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

Accounts Receivable:
Gail Naples *gailn@sys-con.com*

Customer Relations

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

Portable Persistence Using the EJB 3.0 Java Persistence API

The time for standardizing persistent POJOs has come

by Mike Keith and Merrick Schincariol

Experience has taught us that it's not enough to simply have a persistence standard as part of an enterprise specification. It must be a standard that can solve people's problems and be useful to most of the applications that want to use it. While earlier versions of Enterprise JavaBeans (EJB) persistence met some of the needs, they were primarily focused on the distributed problem domain. It is now known, and has been proven by successful commercial products like Oracle TopLink and Open Source projects like JBoss Hibernate, that the objects to be persisted don't have to be anything more than simple Java objects. The proof was in the popularity of these Object-Relational Mapping (ORM) tools; most developers have tended to pick up and use these tools rather than adopt the Java 2 Enterprise Edition (J2EE) entity bean programming standard.

The problem was that even though ORM technology suited them, some corporate IT shops were somewhat uncomfortable with using proprietary APIs in their large-scale applications. They wanted and needed the flexibility and reduced risk that comes with standards-based development. The time for standardization of persistent POJOs (Plain Old Java Objects) had come and the EJB 3.0 Java Persistence API (JPA) was born. It was completed and released as part of Java Enterprise Edition 5 (Java EE 5) in May 2006. Now everybody who has been using a proprietary Java persistence product can develop to a standard set of APIs and benefit from the portability and common experience that standards bring to the table. In this article we will introduce you to a few parts of the JPA API and show how they can enable developers to write portable persistence applications. We will also highlight interesting portability pitfalls that could ensnare the unwary developer.

Mike Keith is an architect for Oracle TopLink and the Oracle OC4J Java EE Container. He was the co-specification lead of EJB 3.0 (JSR 220) and a member of the Java EE 5 expert group (JSR 244) and co-authored (with Merrick) *Pro EJB 3: Java Persistence API*.

The Domain Model

The primary currency of the JPA is the *entity*, which is just a regular Java object that may be persisted to a relational database. Entities can be created, queried for, accessed, modified, and removed from the table or tables that contain the state and are uniquely identified by means of their *persistent identity* or primary key.

Note that entities are quite different from the entity beans of previous EJB versions. Entity beans were full-bodied components that contained built-in remoting, transactional and security logic inserted into container-generated sub-classes by the EJB container at deployment time. Entity beans were created and destined to be entity beans, and were not suited to be anything else. Conversely, entities are simple object classes that don't have to contain any JPA-specific code. A class may often be designated as an entity without even changing a line of code in it; however, if annotations are used, the developer can choose to add them to the code if he decides it's appropriate. The entity object model is completely agnostic not only to the vendor (called the *persistence provider*) but possibly even the fact that it's persistent. For example, a developer could take an existing non-persistent class called Flight and make it persistent simply by indicating that it's an entity through the use of an @Entity annotation in the class:

```
@Entity public class Flight { ... }
```

Or, the developer could leave the class entirely alone and just add an entity entry in a separate XML mapping file:

```
<entity class="Flight">
    ...
</entity>
```

Regardless of which approach is used, the Flight class will continue to function as either a non-persistent class or a persistence entity depending on how it's used. It may have DomesticFlight, InternationalFlight, or other classes that extend it, and these classes may be entities or non-entities, but regardless of the JPA implementation, the object model can be exactly the same. No additional constraints, such as having to implement an interface or extend a special superclass, are imposed on the Flight class by either the API or persistence vendor implementations of the API.

O-R Mapping Metadata

We just saw how either an annotation or XML can be used to designate a class as an entity. This is obviously just scratching the surface of the metadata that is available and that dictates how the entity can be used or mapped. There's a host of JPA metadata in both annotation and XML forms, and the fact that this metadata is defined by the specification is significant for portability.

The largest portion of metadata is typically applied to entities for purposes of object-relational mapping, or mapping the state in the entity to the tables and columns in the database. Until JPA came along every ORM tool had its own way of defining and storing the ORM information for the entities that were mapped to the database. The JPA specification defines specifically and exactly how the metadata should be formed for a group of entities or the entities that make up a persistence unit. For example, we can choose to mark up our Flight class using annotations to indicate how it's mapped to the database.

```
@Entity
public class Flight {
    @Id
    @Column(name="ID")
    int flightNumber;

    @Column(name="DEP_TIME")
    Timestamp departureTime;

    String dest;

    @OneToMany(mappedBy="flight")
    Collection<Passenger> passengerList;

    ...
}
```

The annotations denote how the Flight class is mapped to the database. The @Table annotation is used to indicate to which table the entity is mapped, but it's often not even required since a default table name will be applied in its absence. The default name that is used is the unqualified name of the class. In this case the table will be called FLIGHT. The attribute mappings show how the members of the Flight class correspond to the columns of the FLIGHT table. The @Id annotation indicates that flightNumber is the primary key attribute, and the accompanying @Column annotation shows that it maps to the primary key column ID in the FLIGHT table. Likewise, the @Column annotation on the departureTime attribute denotes that the departure time is stored in the "DEP_TIME" column. Since there's no annotation on the dest attribute, the column name is defaulted to the name of the attribute, or DEST. The passengerList attribute is annotated with an @OneToMany annotation, signifying that it's a one-to-many relationship and the name of the foreign key or join column in the database is specified by the Passenger.flight attribute mapping.

All of this mapping metadata is very convenient in that regardless of the JPA implementation runtime, the same annotation configuration will imply exactly the same OR mappings. Even the same default values will be used since the rules for defaulting are dictated by the specification.

The XML mapping file alternative is equally portable in that the same mapping file(s) can be used when running with any and every compliant JPA vendor. The mapping files may be used to increment or even override the mapping information stored in annotations, and because the overriding characteristics are defined to the level of entity attributes, they can be portably overridden. For example, if we had an XML mapping file that contained the following, it would cause the departure time and destination to be stored in the DEPT_TIME and DST columns, respectively.

```
<entity class="Flight">
  <attributes>
    <basic name="departureTime">
      <column name="DEPT_TIME"/>
    </basic>
    <basic name="dest">
      <column name="DST"/>
    </basic>
  </attributes>
</entity>
```

Persistence Unit Metadata

The other main type of metadata is the persistence unit metadata, or metadata that applies to an entire group or configuration of entities. This metadata is stored in a file called persistence.xml and includes the things that are normally defined for a given runtime environment. For example, although JPA can run in a Java SE environment it will typically be used in an application server. When running in Java EE the persistence provider will have to know which data source to use to connect to in order to retrieve and store entity data. The JNDI name of the data source can be specified in the persistence.xml file, and is portable as long as that data source is configured and present in the JNDI namespace of the server runtime being used.

A number of provider-specific properties can be included in the persistence.xml file. The properties are primarily for non-standard features, but they are specified in a standard way that lets different vendors use the same format. So while developers may need to specify a different property for each vendor, a given vendor will recognize the properties that apply to it but ignore those that it doesn't know about. As an example, if a developer wanted the logging level to be set to the level that included printing the generated SQL and he was running in either TopLink Essentials or Hibernate then he would have the following property section in his persistence.xml file:

```
<properties>
   <property name="toplink.logging.level" value="FINE"/>
   <property name="hibernate.show-sql" value="true"/>
</properties>
```

> "JPA can be used to write applications without being bound to any particular persistence provider or vendor"

Merrick Schincariol is a senior engineer for the Oracle OC4J Java EE Container. He was a lead engineer for Oracle's EJB 3.0 release and co-author of *Pro EJB 3: Java Persistence API*. Before joining Oracle, Merrick developed enterprise and large-scale systems for the telecommunications industry.

Spring 2.0 users can make use of the additional Spring abstractions over some of the more common properties, such as logging, thereby gaining an additional level of portability across vendors.

Persistence Operations

The way to operate on entities in JPA is by invoking a method on an *entity manager* and passing the entity as an argument to the method. The entity manager provides a common interface for entity operations and provides entity management within a transactional or even longer scoped context. For example, to persist a Flight entity, one only has to have access to an entity manager and invoke the persist() method on it, passing in the Flight entity as follows:

```
Timestamp depTime =
    Timestamp.valueOf("2006-09-30 05:07:0");
Flight newFlight =
    new Flight(552, depTime, "San Francisco");
em.persist(newFlight);.
```

When the transaction commits, the entity will be guaranteed to be committed to persistence storage. Likewise, one may use an entity manager to obtain a pre-existing instance of a Flight entity based on its flight id:

```
Flight flight552 = em.find(Flight.class, 552);
```

The entity manager has a complete and understandable API that is the main gateway to using JPA. However, persistence providers implement entity managers, and because allowances are made in the specification for different kinds of implementations, the semantics are sometimes a little looser than what you might expect. For example, in the persist() operation above we mentioned that the entity will be guaranteed to be committed to the database when the transaction commits. We didn't say when the data actually gets written to the database because the specification actually allows the provider either to eagerly write it or defer the write until the transaction commits. The only thing the user can rely on is that by the time the transaction has successfully committed the data will be in the database. It turns out that almost all vendors will actually defer the write because it's more efficient, but users who rely on this fact could be in trouble if they change providers (performance and scalability degradation aside).

The entity manager is a scaled-down version of the session API that has long been used in TopLink or a similar session API in Hibernate. The most common and useful operations on these session APIs have been normalized into the entity manager and represent an API that spans all persistence providers. Programming to the JPA EntityManager API will enable an application to be portable across these providers and prevent non-standard or proprietary features from slipping in.

Queries

To execute a query in JPA a query object must first be obtained from an entity manager. The query criteria is specified either dynamically in code or statically in metadata, and is normally expressed in terms of the JPA query language called the Java Persistence Query Language (JPQL). Queries are executed and depending upon the query the results may be returned either as entities, temporary non-entity objects, or even report data.

JPQL is based on EJB QL but is more powerful and more flexible. It still provides an abstraction language that's used to express queries in terms of entity state and relationships, but is expanded to include a host of new language features including a larger set of functions, outer joins, named parameters, sub-selects, aggregation, bulk updates and deletes, and much more. Because JPQL is a database-neutral language, queries expressed in JPQL are not only portable with respect to persistence providers but also across databases.

Queries may also be created using native SQL. This will typically reduce portability across databases but won't affect persistence provider portability. Queries that use SQL will produce uniform results and are mapped to entities in a standard way. SQL queries are discouraged, however, unless really needed, since they're less maintainable and result in a tighter coupling to the database.

A typical dynamic query to return a list of all of the flights going to a specific destination would be created and executed the following way. The destination is a named parameter that is bound to an argument before being executed, so the same query instance can be reused for querying different destinations. We'll find all the flights going to San Francisco.

```
Query q = em.createQuery(
    "SELECT f FROM Flight f WHERE f.dest=:destination");
q.setParameter("destination", "San Francisco");
List sfResults = q.getResultList();
```

While this query is completely portable in terms of its execution, what about the semantics of the query, or the results that are returned? Could the results differ depending upon the context in which it's executed? For example, if there was a transaction in progress and a new flight to San Francisco was added in the transaction, is that flight going to be returned by all providers? Remember that if the transaction hasn't been committed yet, depending

> "The query criteria is specified either dynamically in code or statically in metadata, and is normally expressed in terms of the JPA query language called the Java Persistence Query Language (JPQL)"

upon the implementation, the new flight may not even have been written to the database.

The answer is that there is something called a *flush mode* that determines whether all changes in the transaction have been written out. By ensuring that the flush mode setting causes a flush to occur before transactional queries are executed, the results will always be the same regardless of the persistence implementation. In fact, to achieve portability and query results that most people would expect, the flush mode is set this way by default. To avoid the performance overhead of issuing a flush before a query, the flush mode is often changed. However, to ensure portability this should only be done when it's known that any entities modified earlier in the transaction won't affect the outcome of the query. Alternatively, queries can be executed outside of transactions, typically improving the performance of the query in the process.

One of the best practices for creating queries is to define them statically using the *named query* facility. Named queries are queries that are defined in annotations or XML and offer an additional form of application portability. The query criteria may be separated from the application code and filled in using JPQL, SQL, or any proprietary query language, such as the TopLink expression framework. The above query could be defined as a named query by defining the following annotation that specifies the name of the query and the query criteria as follows:

```
@NamedQuery(name="Flight.findByDestination",
  query="SELECT f FROM Flight f WHERE f.dest=:destination")
```

The query is invoked by obtaining an instance of the named query, binding the parameter, then executing it, similar to the dynamic query example above:

```
Query q = em.createQuery("Flight.findBy
Destination");
q.setParameter("destination", "San Francisco");
List sfResults = q.getResultList();
```

Vendor-specific Features

It's not unusual that a large application has requirements that the JPA 1.0 can't fulfill. After all, the first release of JPA included a lot of features, but as mentioned above, not everything was added. There are still a number of features up for discussion and possible inclusion in subsequent JPA releases that at this point in time are vendor-specific. Pessimistic locking is an example of a feature that's not usually required but on rare occasions is absolutely necessary.

Vendor-specific features can be accessed a number of different ways; some of them better than others. As we saw in the persistence unit metadata section, JPA does provide some mechanisms for vendors to incorporate additional features using standard APIs and the persistence properties are one such way. Query hints are also a way for additional query features to be accessed either programmatically or through metadata. Vendors can define their own query hints that users can add to named or dynamic queries before they're executed.

Additional XML files and annotations are another common way for users to add vendor-specific metadata. Proprietary annotations tend to be more dangerous than XML because they introduce compile-time dependencies in addition to any runtime dependencies that may exist.

In code the vendor-specific EntityManager implementation class can be retrieved by calling a special method on EntityManager and casting the result. A Query can also be cast to a proprietary interface or class. These practices should be used with care because they introduce code dependencies into the application.

Regardless of how the feature is used, the best way to organize vendor-specific feature use is to try to localize it to specific metadata and code areas. This way if porting is required it's easy to find the metadata or code that might need to change.

Conclusion

We've looked at only a few of the features of the EJB 3.0 Java Persistence API, but we have already seen how it can provide applications with a modern and portable platform for object-relational mapping and persistence. By combining the principal and most significant features of the major persistence solutions on the market and in the public domain, JPA can be used to write applications without being bound to any particular persistence provider or vendor. The underlying persistence implementation could be changed with few or no changes to the application code that uses it.

We did see a few examples of how some of the implementations may differ from each other though, so application developers should still be alert to the nuances of the provider implementation. A simple understanding of the API is all that's needed to develop simple persistence applications. However, to develop complex portable applications, a more thorough understanding is critical. For in-depth coverage on the features in JPA, as well as the portability issues of the API, we refer the reader to *Pro EJB 3: Java Persistence API*.

"We've looked at only a few of the features of the EJB 3.0 Java Persistence API, but we have already seen how it can provide applications with a modern and portable platform for object-relational mapping and persistence"

top **MISCONCEPTIONS** that drive

Meet the most misunderstood developer team in the world.

our Crystal Reports dev team crazy

Crystal Reports® is too expensive. Actually, the developer edition is just $595[1] USD (or upgrade for only $315[1]). Complimentary Crystal Assist support[2] provided with purchase.

Crystal Reports doesn't include a free runtime license. Not true, the developer edition includes a free runtime license[3] for each component engine.

Getting reports on the web is complex. False, the developer edition includes crystalreports.com[4] and Crystal Reports Server[5] to speed and simplify web reporting deployments.

Crystal Reports only works in Windows®. Not quite, whether you need to create or deploy reports on Windows, Linux or Unix, we have a Crystal Reports technology for you.

Find out more at: www.businessobjects.com/devxi/misunderstood

Business Objects™

1 Suggested retail price. 2 Complimentary access to support engineers and self-help. 3 Includes an unlimited runtime license for internal use of .NET, Java, and COM engines. 4 Includes ten named user licenses. 5 Includes five named user licenses.

Building a Simple VocabBuilder Application

Creating a good impression

by Kanchan Waikar

To make a good impression, one needs to have a good vocabulary. Management Professionals, University Professors, or GRE/GMAT aspirants – we all benefit from a decent set of words in this competitive world. There are different ways we can improve our vocabulary, such as reading novels, articles, dictionaries and so on, but we often find very little time to do so.

During our "down-time" – when we are stuck in traffic for hours, or standing in some long queue - our mobile phone is often our only source of entertainment. If our mobiles – which one in three people already carry – had word-tutors, this time could be efficiently used to learn new adjectives, verbs and nouns.

To build such an application, words need to be stored in a database. Since simple database implementation (RMS) is dynamic, the database needs to be built on every new mobile, after application is installed. To remove this overhead, the only way to maintain the word-list is in a file. Because of some programming related issues, j2me does not provide a full fledged API for file access. Still, there are ways to use a file in a j2me application. In a Vocab Builder application, words and their meanings, stored in those files randomly, can be retrieved.

There are "n" number of constraints with mobile programming - such as power, speed, UI size, memory size limitations, and many others - which make programming for mobile applications difficult. Probably, this is the reason why we see very few attempts to create such software. However, a simple vocabulary builder can be implemented without much effort. The following article can be used as a guideline to develop a simple vocabulary builder for your mobile phone.

Kanchan Waikar is a software professional working with one of the multinational IT companies. She is very much inclined towards mobile Programming. Currently she is looking for some challenging work on J2me platform.

waikar.kanchan@gmail.com

First, let's look at Implementation Diagram, as shown in Figure 1.

All classes from the above diagram are explained in this article. Let's start with the MIDlet class. Our application will have a MIDlet that will initiate the application.

VocabBuilder Class

Let's define this MIDlet Class and name it VocabBuilder (see Figure 2).

```
public class VocabBuilder extends MIDlet
implements CommandListener {
```

This VocabBuilder MIDlet displays the operations menu for the user. The menu provides different functions that are supported by the VocabBuilder tool. Since the tool has 5 functionalities, the same are provided in the operations menu.

```
menuList = new List("Expert Vocab
Building",List.IMPLICIT);
menuList.append("Learn Adjectives",adjecti
vesImage);          menuList.append("Learn
Verbs", adjectivesImage);
menuList.append("Learn Nouns", adjectivesIm-
age);
menuList.append("Add words", verbsImage);
```

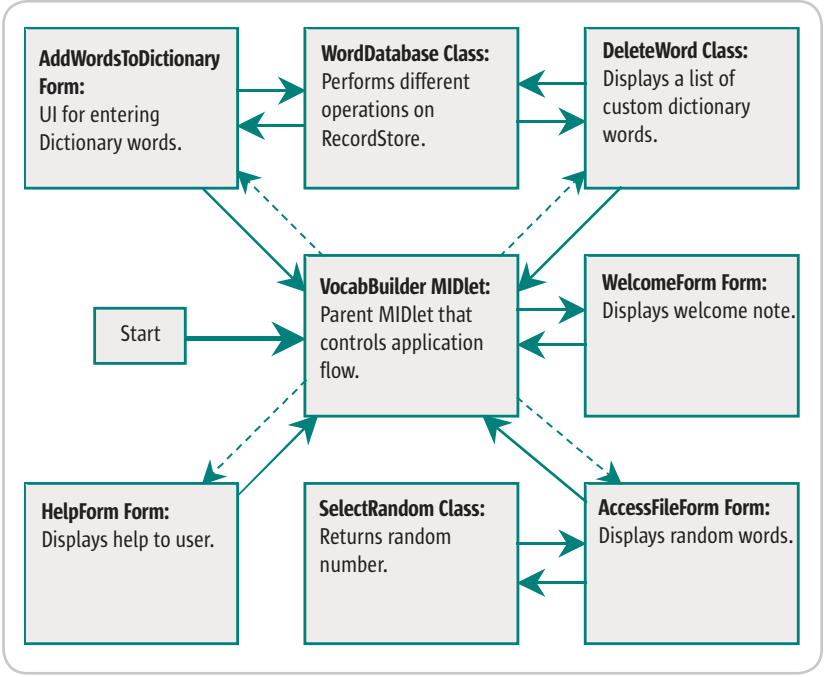
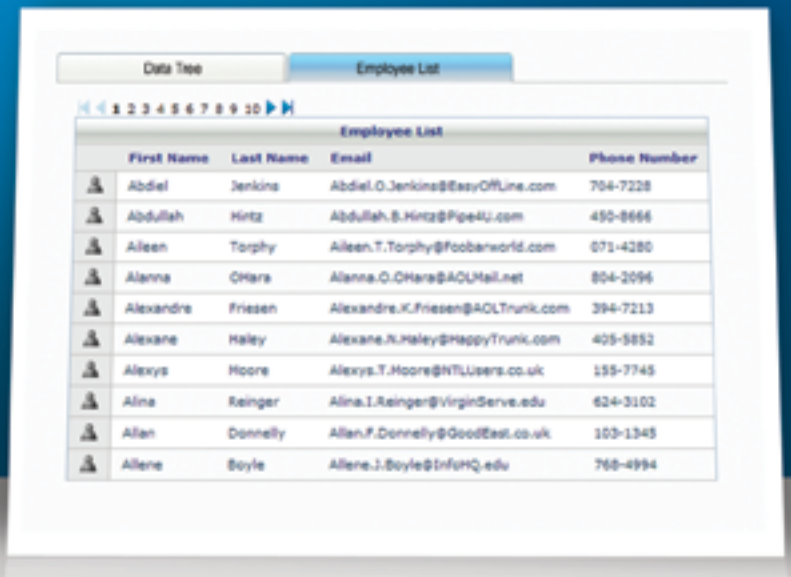


Figure 1 Implementation diagram– *Plain arrow*: Normal Flow; *Dashed arrow*: Conditional flow

Speed.
Simplicity.
Style.

Build a better User Interface with NetAdvantage



NetAdvantage® for JSF

2006 Volume 1

AJAX-enabled JavaServer™ Faces components

Speed – Built to support large, data-driven applications

Simplicity – Intuitive features for end users; simple tag interface for developers

Style – Fully customizable look & feel

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 navigation menus listbars trees tabs explorer bars editors

Figure 2

Figure 3

Figure 4

```
menuList.append("View / Delete words", verb-
sImage);
menuList.append("Help ",helpImage);
```

To provide the word-list, a file is used as a datastore. In this class, a string that contains the name of this file stores different words and their meanings. Providing the get() method for this variable will ensure that the get() method returns the name of the file currently in use by the application.

```
private String wordDatabaseFile;
```

As shown in Figure 3, the menu UI is displayed after the displayList() function of the VocabBuilder class is called on.

After the user selects a particular option from List, the corresponding operation is performed by implementing the CommandListener Interface. In need of a common algorithm to implement functionality of the first three options, we call on the method of getRandomWords() of the accessFile class. This class will, in turn, fetch the words and display them on the Form. But before calling on the aforementioned method, we need to set the value for the wordDatabaseFile string.

```
else if (c == List.SELECT_COMMAND)
       {
         int selection=menuList.getSelect-
edIndex();
          switch(selection)
          {
            case 0:
                        wordDatabaseFile =
"adjectives.kw";
            accessFile.getRandomWords();
            break;
        case 1:
    wordDatabaseFile = "verbs.kw";
    accessFile.getRandomWords();
    break;
        case 2:
            wordDatabaseFile = "nouns.kw";
             accessFile.getRandomWords();
            break;
        case 3:
             AddWordsToDictionary awdForm =
new
                          AddWordsToDictio
nary(this);
                  Display.getDisplay(this).
setCurrent
                              (awdForm);
         awdForm = null
            break;
        case 4:
            DeleteWord dw  = new
DeleteWord(this);
    dw = null;
            break;
        case 5:
             HelpForm helpForm = new
HelpForm(this);
          Display.getDisplay(this).setCurrent
                              (helpForm);
            helpForm = null;
            break;
          }
         }
```

For the last three options, different classes are intitialized. These classes, in turn, will provide the functionality for adding, updating, deleting, and viewing the custom Dictionary.

AccessFile Class

This class fetches three random words at a time from the selected file as shown in Figure 4. In order to display these words, this class needs to extend Form Class.

```
public class AccessFile extends Form imple-
ments CommandListener
```

Note: Generally, when people read words that are consecutive, they tend to get bored. Hence, the random words generation strategy can be used to retain the interest of the user.

We need a "Next" button in order to display the next set of random words. This application opens the file in which word-meaning pairs are stored. We all have come across the apothegm, "differ-

> "Generally when we read words that are consecutive, we tend to get bored. Hence random words generation strategy was used to retain the interest of the user"

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

Java on the desktop is back!

RCP Developer™ version 2.0

eclipse READY™

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

STAND ON THE SHOULDERS OF GIANTS

Create blazing fast rich Java clients.

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

Build rich Java client applications faster.

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

Deliver a more consistent user experience.

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

Fully exploit the Eclipse Rich Client Platform.

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

—Mike Milinkovich, executive director of the Eclipse Foundation

RCP Developer™ 2.0

SWT Designer™

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

WindowTester™

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

Help Composer™

Document. Quickly create documentation in the Eclipse Help system, organizing and formatting Help files; automatically translate Help files into Web pages for your public Web site.

RCP Packager™

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

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

instantiations
the development tools experts

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

The leader in Eclipse RCP development tools

eclipse FOUNDATION MEMBER™

Figure 5

Figure 6

ent people, different needs". Accordingly, the user can choose a "learning area" appropriate to his or her needs. Some people are weak in adjectives, whereas others are weak in nouns, so different interfaces are provided for nouns, adjectives and verbs. Since implementation of all three is identical, an AccessFile class was created. To open the file and read the contents, the following function is used:

```
inputStream =
      getClass().getResourceAsStream(vb.get-
DataFileName());
```

inputStream is an object of InputStream class. Using getClass.getResourceAsStream(), we can open the file that is stored in the "res" library of the application package. After opening the file, we need to position the cursor in a random position. This can be done using following function:

```
inputStream.skip(selectRandom.getRandomPosi-
tion())
```

Here, a selectRandom class is used, which returns the random position in the file. By using the skip function of inputStream object, we reposition the cursor in a random position.

After a random position has been selected, the selected words can be fetched. Take extra care in the format in which the selected file is stored. Since the file used in the application has a word-meaning pair in a line the following strategy is used for fetching:

```
 for(int j = 0; j<3;j++)
    {
 /** Create the word by adding all the char-
acters */
     while ((tempCharRead = inputStream.
read()) != -1 &&
              tempCharRead != '\n' )
     {
     word_meaning[j] = word_meaning[j]+
(char)
                        tempCharRead;
     }
     this.append(wordMeaning);
    }
```

Since a word and its meaning are separated by a colon, the pair can be displayed in the same way on Form, using the append() method. In formatting, string item data type can be used. Next and Exit buttons are provided on the form for navigating through the word list.

```
    if (cmd == nextCommand)
  getRandomWords();
  else if (cmd == exitCommand)
      vb.displayList(); // This is the main
MIDlet Operations List.
```

If the user presses the next button, the application displays a new set of words, whereas if Exit is selected, the application takes control to the main List.

SelectRandom Class

This class fetches the file and depending on its size, it generates a random value. If the value generated is positioned towards last word, then the application might display only one or two words instead of three. In order to avoid this the maximum value to be generated can be set:

```
int skipValue=  Math.abs((int) System.cur-
rentTimeMillis() % count);
```

The maximum feasible number that will prevent the display of less than three randomly generated words is "count". A more sophisticated algorithm can be used for generating the random number.

```
AddWordsToDictionary Class
```

Along with rapid learning mode, a customized dictionary can be provided to the user. An interface that will let the user maintain his dictionary is needed. In order to make the application compatible with a wider range of handsets, the architecture of the application needs to be as simple as possible. The database package can be used for better management, but only highly sophisticated handsets can support the Database package. Therefore, the RecordManagement Structure is needed to create, close, delete, and rebuild the database. It is important to note that the record structure doesn't support the primary key concept, so certain checks need to be implemented in the component to make it support primary key concept.

Now, let's look at the structure of the AddWordsToDictionary Class. Below (See Figure 5) is the UI that takes the user's words and stores them in the dictionary.

By providing "My dictionary", the user can maintain his or her own dictionary.

"Along with rapid learning mode, a customized dictionary can be provided to the user."

MAKE ANSWERS TO PERFORMANCE PROBLEMS COME TO YOU.

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

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

PANORAMA

OPNET Panorama
Real-Time Application Analytics

www.opnet.com/pinpoint

OPNET®
Making Networks and Applications Perform™

OPNET Technologies, Inc. 7255 Woodmont Avenue, Bethesda, Maryland 20814 phone: (240) 497-3000 • e-mail: info@opnet.com • NASDAQ: OPNT

In order to support this functionality, the following operations are needed:

1. Insert
2. Update (here Appending new meaning will be more useful)
3. Delete (optional, but can be useful if the user is planning to have a new set of words in his dictionary everyday)
4. View Dictionary

This class implements the first two functionalities. A Form is needed to display TextFields, from which the user can enter the data.

```
  wordTextField= new TextField("Enter Word",
"",20,
                                TextField.
INITIAL_CAPS_WORD);
  meaningTextField= new TextField("Enter
Meaning","",50,
                                TextField.
INITIAL_CAPS_WORD);
```

Since user-input is being handled, extra care needs to be taken in entering data into the database to ensure proper validations are performed. Numbers are not allowed in word or meaning value.

In order to perform the action, depending on the command entered by the user, a CommandListener Interface needs to be implemented.

```
public void commandAction(Command command,
Displayable displayable)

After the user has entered the word and its
meaning, the word needs to be entered into
the dictionary.

if(command==OkCommand)
{
 String wordField=wordTextField.getString();
 String meaningField=meaningTextField.get-
String();
       // Perform null input validations.
 if(wordDatabase.wordExists(wordField))
    // append new meaning
 else if(wordDatabase.getNumberOfRecords()+1
>
                                MAX_WORDS)
   // Display Database Overflow Alert
```

Figure 7

```
 else //If everything is fine
  wordDatabase.writeRecord(wordField,meaning
Field);
 // Display Acknowledgement Alert
```

This class appends the new meaning to the word if the word already exists in Database.

DeleteWord Class

This interface can be used in order to view the dictionary (See Figure 6). Since the word-meaning pairs in the list are being shown, a List needs to be declared.

The following code is used to fetch the words from the database and create a list:

```
RecordEnumeration wordRecEnum =
        wordDBReference.
enumerateRecords(null, null,true);
RecordEnumeration meaningRecEnum =
        meaningDBDatabaseReference.
enumerateRecords(null, null, true);
while (wordRecEnum.hasNextElement())
{
    String fetchedWord =
      new string(wordRecEnum.nextRecord());
    String fetchedMeaning =
      new String(meaningRecEnum.nextRe-
cord());
    displayList.append(fetchedWord+" :
                        "+fetchedMeaning,
bullet);
        }
```

If the user selects a word to delete, then he or she needs to be shown the confirmation Alert message. If the user selects "Yes", then the Record can be deleted and the information can be displayed. The following code will delete the selected word from the database:

```
wordDatabase.deleteScripRecord(selection.
substring
(0,selection.indexOf(" ",2)));
 Alert recordsAlert = new Alert("Deletion
Successful", "  Word Deleted !!       ",
alertImage,AlertType.ERROR);
```

Helpclass

This class displays static information about the application's functionalities as shown in Figure 7.

Conclusion

This article demonstrates the implementation of a Simple vocabBuilder application. The application shows random adjectives/verbs/nouns depending on the user input. It also allows users to maintain their own dictionary and provides add/delete/update/view modes on a custom dictionary. There is great scope for upgrading this application to a much more sophisticated version. This enhanced version will be compatible with only recent, well-groomed handsets. An extensive database structure can be implemented in order to add Artificial Intelligence to the application. A search functionality can be implemented to search the database and return the meaning of the word entered. Since we have used a sequential file data structure for storing the words and thier meanings, a search would have consumed a greater amount of processing time. Hence, this functionality was not implemented. Using the database / RecordStore Structure, this functionality can be easily implemented by firing a SQL Statement, or by doing comparisons in case of RecordStores. Further, we can also provide some word usage statements along with antonyms and synonyms. For audio-learners, the Audio Play feature could be added to Rapid Learning mode.

"People say that you learn fast when you hear the words said hence Audio Play feature could be added to Rapid Learning mode"

Only JReport Delivers 100% Java Embedded Reporting

What a mess! Especially when you try to integrate old Client/Server technology into your Java environment.

Only JReport delivers seamless integration with your Java EE application architecture – which helps reduce development costs, increase developer productivity and provide scalable access to operational data across the enterprise. After all, isn't that the point of reporting? JReport is the most flexible, scalable 100% Java embedded reporting solution available today.

Avoid being burned by old Client/Server technology. Learn how to simplify your reporting solutions with JReport

Download the JReport Embedded Reporting Kit or call 240.477.1000 now.

JReport®
JINFONET SOFTWARE

www.jinfonet.com/JavaReporting

AOP, IoC, and OO Design Patterns Without Frameworks

Finding the right balance

by Jinsong Yang

I'd like to share some of the design highlights of a large-scale content distributing system I worked on a while back. Some of the highlights may seem trivial; some may be a little more complicated. To me, software design is a matter of finding a balance between applying available technologies and fulfilling real-world requirements and constraints. The goal of design is always to ensure both the runtime and development-time quality of the software.

I'll be using two of the components from the project, Scheduling Management Component (SMC) and the Data Access Layer (DAL), for purposes of illustration. The function of SMC is to monitor the database and detect any newly added or updated content, and schedule that content for delivery based on some business logics. The DAL, as its name implies, provides data access services.

Extensibility by OOP Patterns and Principles

In the early design phase of SMC, we quickly lay out some of the candidate classes, as well as the relationships among them. In UML notation, class details are omitted at this stage. Figure 1 shows some of the major classes. The classes are by no means final, and neither are the relationships. We're trying, however, to capture the business requirements as much as possible.

We've designed the ScheduleManager class as the façade of this component. It exposes external APIs. Other components that need to interact with SMC do so through this class. ScheduleManager is also in charge of managing the lifecycles of classes inside SMC.

One of the business requirements is that the first production release of the software won't support clustered deployment, but it will in later releases. One of the keys to supporting clustering is managing the states of the component instances deployed across the clustered nodes so that they're always in sync. Keeping this in mind, we decide to centralize the cluster-sensitive states of all the SMC classes into one class rather than having each class manage its own states. This results in a SchedulingContext class. This class knows how to save and retrieve various states of the component. When the time to support the cluster comes along, instead of having to open up each class and make changes, all we have to do is change the SchedulingContext class – that modifyies the way this class accesses states so that all the clustered instances share the same states, virtually or physically.

The UML diagram in Figure 2 shows some major classes after introducing the SchedulingContext class.

One issue we immediately notice with this design is the tight coupling between the SchedulingContext class and the classes that depend on it. As you can see, quite a number of classes depend on the SchedulingContext class. If we have to change the SchedulingContext class for any reasons (fixing bugs, adding new business features, switching to other application server, etc.), chances are we also have to make changes to the dependent classes.

Following OO best practices, interfaces are good at promoting loose coupling. In fact, it's always a good idea to code to interfaces rather than concrete classes (another simple yet powerful OO principle). The solution is simple

The Open Close Principle

In the OO world, there's this design principle called the Open Close Principle. It says that software should be designed so it's open to extension and closed to modification. Set aside its fancy name, it basically suggests that software should be extensible function-wise (i.e., open to extension) without having to open up existing code and modify it (i.e., closed to modification). Functional extension should be done by creating new modules and plugging them into the existing system.

The consequence is beneficial. After all, you don't want to modify your well-tested code and risk your code to new bugs, which can result in "wreck-a-mole"-style bug fixing (fixing one bug introduces more).

Jinsong Yang is a senior application engineer in a leading digital content providing company. He has devoted the last five years to designing and developing large scale J2EE applications. He holds an MS in Computer Science from UCLA.

yang.jinsong@gmail.com

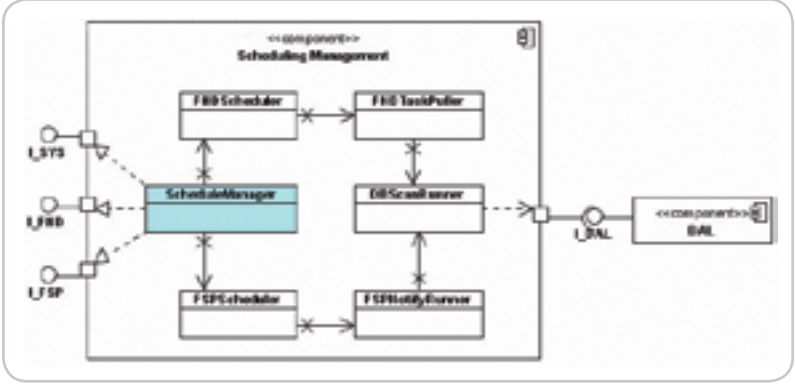


Figure 1 The Scheduling Management Component (SMC) class diagram

enough – we abstract the scheduling context by making SchedulingContext an interface. We also want this interface's client to be unaware of the actual implementation, and the GoF Abstract Factory Pattern does the job.

The design of the Scheduling Context is shown in Figure 3. When it comes time to support clustering, we can do it with little programming. It makes it a lot easier to maintain the software after it goes into production. We also avoid vendor lock-in by shielding the cluster-sensitive implementation, which is likely to be vendor-specific.

Another consequence of moving the states of an individual class into SchedulingContext is that we can now design other stateless classes, which is good because:

- A stateless class is thread-safe, avoiding potential issues in a multithreading environment (which SMC is)
- Generally speaking, stateless objects scale better
- Stateless classes are easier to maintain

Note that, at this point, ScheduleManager and SchedulingContent are acting together as the "container" of the other classes in the sense that:

- The ScheduleManager class intercepts all client calls to the component
- The ScheduleManager class manages the lifecycles of other classes
- The SchedulingContext interface provides a gateway to access component states

This is intended and, as you'll see later, we'll use the container-like feature of these two classes more in our design.

Decoupling with IoC Pattern

So far, everything is pretty simple and straightforward. What else can we do to make the design better?

With the current design, unit testing isn't a trivial task. Let's take the FNDTaskPuller class as an example. Listing 1 shows the simplified version of this class. For demonstration purposes, let's assume this class has a business method, someBizMethod().

To test this class, we'd create a test class similar to the one shown in Listing 2.

What's the problem with this code? We're testing SchedulingContextSimpleImpl (which is the concrete class of SchedulingContext) indirectly. But we really mean to test the FNDTaskPuller class. This isn't right. As we all know a unit test should never go outside of its own class boundary.

Furthermore, it's hard to control the states of the SchedulingContextSimpleImpl in order to test the different behaviors of FNDTaskPuller class.

In practice, a common technique to overcome this kind of tight coupling is using mock objects, which can assist in separating unit tests. Mock objects themselves, however, require extra coding efforts. This extra coding effort can be significant, buggy, and cause maintenance problems. What more? Developers have to replace mock objects with real classes at deployment.

The reason for this problem is the way we acquire the reference to the SchedulingContext object in the FNDTaskPuller class. In this case, the FNDTaskPuller class is asking for a reference to a SchedulingContext object. Explicitly.

To get around this, we need to change the way we obtain an object reference. This is where Inversion of Control (IoC) comes into play.

With IoC, objects obtain references to their dependent objects passively. The IoC container literally "injects" the dependency into the classes.

Now, we need an IoC container for our design to wire the objects. We have the choice of using an existing container product or building our own. Normally in-house framework building is considered a bad practice because of its complexity and inefficiency. However, we decided to do it anyway after carefully figuring out what we really needed. Some of the reasons are:

- We don't intend to build a framework that supports complete IoC features. Rather, our goal is to provide a feasible solution to our immediate problem: loosening up the coupling between SchedulingContext and other classes. At this point, some simple dependency wiring functions will do the trick.
- The team didn't have enough experience with third-party containers when we started development, and the project timeline was a tight learning curve.
- The team had to use some legacy code and worried that it would require some architectural modifications to integrate third-party containers with the legacy codes.
- It's a good starting point for developers to get their feet wet on IoC concepts and implementation.

Our solution was to embed the simple dependency wiring functions in the ScheduleManager class, which, as mentioned, was already acting as the SMC "container." It also made sense to embed the IoC functions in

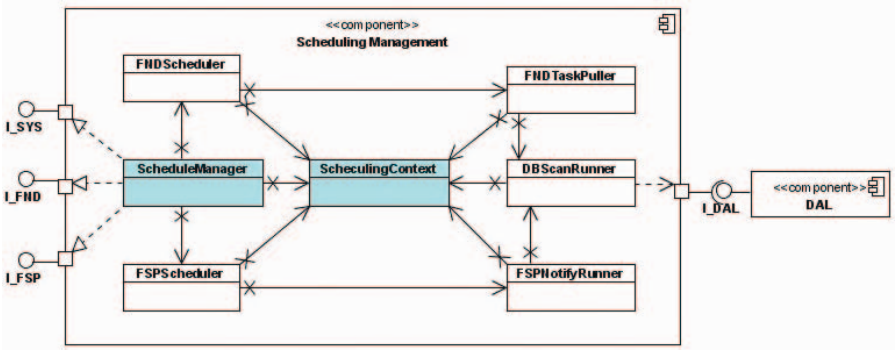


Figure 2 All classes have dependencies on the SchedulingContext class

Inversion of Control

Inversion of Control (IoC), also referred as Dependency Injection (DI), is a powerful pattern that can be applied in software design to reduce coupling between components. It's a key feature in lots of lightweight containers, which help assemble components from different sources into a cohesive application.

IoC comes in three flavors – type 1, type 2, and type 3 IoCs; however, they're more often referred to by their more descriptive names: interface inject, setter injection, and constructor injection, respectively.

Nowadays many IoC container products exist such as Spring and PicoContainer.

The Abstract Factory Pattern

The intent of the Abstract Factory Pattern is to provide an interface for creating families of related or dependent objects without specifying their concrete classes. The benefits of this pattern are that it isolates clients from concrete implementation classes, makes exchanging product families easy, and enforces the use of products from only one family.

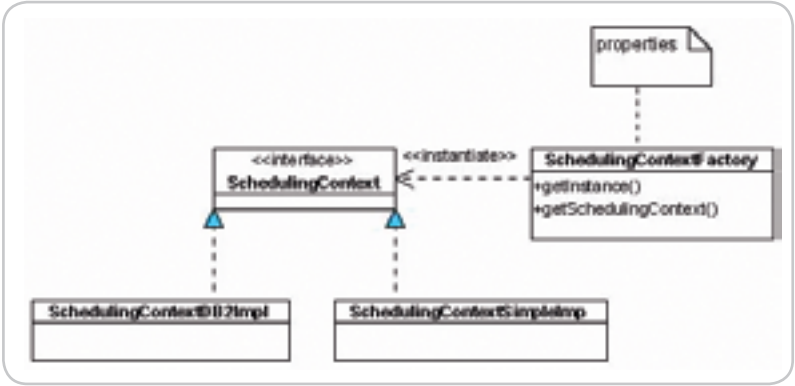


Figure 3 Scheduling Context UML class diagram

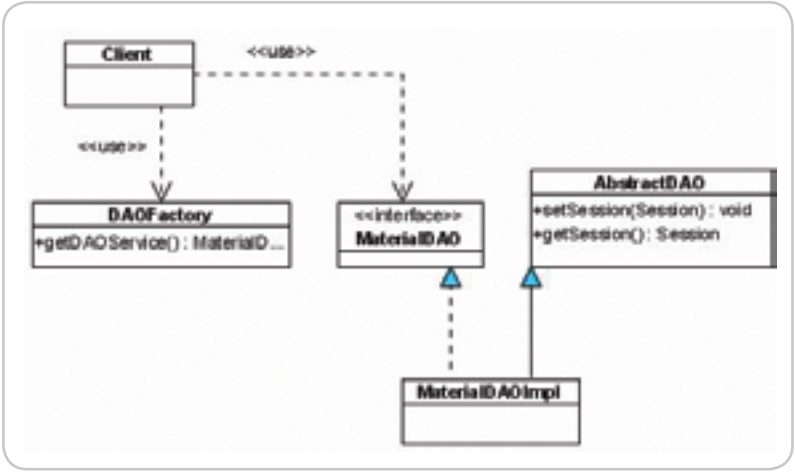


Figure 4 AOP class diagram

Aspect-Oriented Programming

Aspect-Oriented Programming (AOP) attempts to aid programmers in the separation of concerns, specifically cross-cutting concerns, as an advance in modularization. The idea is to encapsulate concerns into separate features and minimize their functional overlaps as much as possible.

Compared to procedural programming, OOP methodologies take a significant step towards separation of concerns.But there are concerns when OOP fails to separate, one of which is a concern that cuts across many modules of an application (hence the name cross-cutting concerns).

The following definitions are based on Wikipedia (http://www.wikipedia.org/).

Advice: describes a certain function, method, or procedure that is to be applied at a given join point of a program.

Join point: a point in the flow of a program.

Pointcut: a set of join points. Whenever the program execution reaches one of the join points described in the pointcut, the advice associated with the pointcut is executed.

Java is full of frameworks, AOP included: : AspectJ, Spring, JBossAOP, etc.

this class. Because of the simplicity of construction injection, we refactored all the classes depending on the SchedulingContext so that they were ready for constructor injection. Listing 3 gives the FNDTaskPuller class as an example.

All we have to do in the ScheduleManager class is to instantiate a SchedulingContext object and assign it to classes that need a reference to it. If, some time in the future, a full IoC container is needed, we can just modify the ScheduleManager class.

Taking Care of Scattered Code

While designing the data access layer, we foresaw that there would be a lot of scattered "plumping" code – code that doesn't do anything related to business functions. They are only here to fulfill middleware functions: JNDI lookup, JDBC resource management, exception handling, etc. This code will spread into almost all data access objects (DAOs) and, in most cases, this code is identical method to method. We all hate scattered code because it's a maintenance nightmare, is error-prone, and makes code hard to understand.

List 2 is an example of a how we would have implemented our data access methods without AOP.

In Listing 4, only one line of the code is actually "doing something." The rest is just "plumping" code. When "plumping" code starts leaking into your application, chances are you'll find yourself hunting down all the application code when requirements change.

There's one thing we can do. While OOP makes software design modular, AOP makes code modular. And modularity is good.

Once again, we faced the choice of using an existing AOP framework or building our own. We decided not to use any of these frameworks for the same reason why we didn't use a IoC container. Instead, we separate the concerns programmatically in our code. Although this is not the most elegant, cleanest way, it's the fastest, which, in our case, is a big gain.

To separate the cross-cutting concerns from the DAOs, we came up with the following class design:

- DAO classes. These are classes that implement data access functions. There can be an arbitrary number of DAO classes, and they're POJOs.
- DAOFactory is the advice class. It implements the cross-cutting concerns. It also defines pointcuts, in our case, all DAO methods. It does so by handing out proxy objects of the corresponding DAO classes.

Figure 4 shows the class diagram. A client asks DAOFactory for a service provided by a particular DAO by passing in the DAO class name. Upon request, DAOFactory instantiates a proxy instance and a real instance of that class, and hands the proxy back. The client then makes calls on the proxy object, instead of the real instance of the DAO class, to consume the service.

Because all calls are made through the proxy object, the proxy can intercept the calls, wrapping the business methods with cross-cutting concerns. Listings 5, 6, 7, 8, and 9 show a simplified version of these classes. Note that MaterialDAO is just an example of many DAOs.

Note that with this design, the DAOs aren't completely POJOs: Each DAO has to provide an interface and extend the AbstractDAO super-class. The clients are also aware of the concrete implementation class. With this solution, however, we're trying to land somewhere between the pain of living with scattered code and the burden of implementing a complete AOP framework.

Conclusion

In this article, I've just covered a small number of the design issues in our project. It's an even smaller part compared to the real world of software design. The idea is to present our way of thinking of design. We believe this kind of thinking will definitely familiarize the team with IoC and AOP concepts and prepare it for next step forward.

Listing 1: FNDTaskPuller class

```
public class FNDTaskPuller {
    private SchedulingContext ctx;
    // Other code

    public FNDTaskPuller () {
        this.ctx = SchedulingCtxFactory.getInstance().getSchedulingCon-
text();
        // Other code
    }

    public boolean someBizMethod() {
        // Method implementation
    }

    // Other code
}
```

Listing 2: FNDTaskPullerTest class

```
public void testSomeBizMethod () {
    FNDTaskPuller puller = new FNDTaskPuller();
    boolean result = puller.someBizMethod();

    // assertions goes after
    assertTrue(result);
    assertEquals(…);
}
```

Listing 3: CDBTaskPuller class gets ready for constructor injection

```
public class FNDTaskPuller {
    private SchedulingContext ctx;
    // Other code

    public void FNDTaskPuller (SchedulingContext ctx) {
        this.ctx = ctx;
    }

    // Other code
}
```

Listing 4: DAO method implementation without AOP

```
public void updateFNDData(FNDDataVO data) {
    Session session = null;
    Transaction tx = null;
    DataVO data = null;

    try {
        session = sessionFactory.openSession();
        tx = session.beginTransaction();

        session.update(FNDDataVO.class, data);

        session.flush();
        tx.commit();
    }
    catch (HibernateException e) {
        logger.debug("error updating FND data.", e);
        if (tx != null) tx.rollback();
    }
    finally {
        if (session != null) session.close();
    }
}
```

Listing 5: Example DAOFactory class

```
public class DAOFactory {
    // Code omitted here

    /**
     * Create the proxy instance based on the class name
     */
    public Object getDAOService(String daoClassName) {
        Object proxy = null;
        try {
            Class c = Class.forName(daoClassName);
            InvocationHandler h = new HibernateInvocationHandler(daoCla
ssName);
            proxy = Proxy.newProxyInstance(c.getClassLoader(),
                                           c.getInterfaces(), h);
        }
        catch (ClassNotFoundException e) {
            logger.error("Error", e);
        }
        return proxy;
    }

    public class HibernateInvocationHandler implements InvocationHandler
{
        private String daoClassName;
        public HibernateInvocationHandler(String daoClassName) {
            this.daoClassName = daoClassName;
        }

        public Object invoke(Object proxy, Method method, Object[] args)
            throws Throwable {

            Class c = Class.forName(daoClassName);
            AbstractDAO target = (AbstractDAO)c.newInstance();

            Session session = null;
            Transaction tx = null;
            Object result = null;
            try {
                session = sessionFactory.openSession();
                tx = session.beginTransaction();
                target.setSession(session);

                result = method.invoke(target, args);

                session.flush();
                tx.commit();
            }
            catch (HibernateException e) {
                logger.error("Error. Transaction rolled back.", e);
                tx.rollback();
            }
            finally {
                if (session != null) session.close();
            }

            return result;
        }
    }
}
```

Listing 6: Example AbstractDAO class

```
public class AbstractDAO {
    private Session session;

    public void setSession(Session s) {
        this.session = s;
    }

    public Session getSession() {
        return this.session;
    }
}
```

Listing 7: Example DAO Interface

```
public interface MaterialDAO {
    public MaterialVO load(String id);
}
```

Listing 8: Example DAO implementation

```
public class MaterialDAOImpl extends AbstractDAO implements MaterialDAO {
    public MaterialVO load(String id) {
        return (MaterialVO)this.getSession().load(MaterialVO.class, id);
    }
}
```

Listing 9 Example client code

```
DAOFactory f = new DAOFactory();
MaterialDAO dao = (IMaterialDAO)f.getDAOService("MaterialDAOImpl");
MaterialVO material = dao.load("3");
```

NetBeans

Interview with Tim Cramer, executive director of tools at Sun

by Joe Winchester

Recently I was able to talk to Tim Cramer, executive director of tools at Sun, about NetBeans. Tim started in engineering doing supercomputer compiler work, moved to more generalized hardware compiler work, and naturally moved to JIT/dynamic compilers in Java during its first few years. Tim's first management job was in the Java performance group, working to improve the base performance of Java SE and EE. He followed as the director of NetBeans in August of 2004 and is now the executive director for all Java tools at Sun.

Tim Cramer
Sun Microsystems

There seems to have been a lot of activity around NetBeans' 5.0 release lately. What's all the excitement about?

Tim Cramer: NetBeans 5.0 delivered a lot of new features that blew the Java developer away: the GUI Builder formerly known as Project Matisse, the code-aware collaboration tools, the NetBeans Profiler, and Web framework support. In addition, we made improvements to existing features: better CVS support, improvements in the Java editor, etc. It's a pretty compelling product release. With NetBeans 5.0 we really reached a tipping point in the market: more and more users and more and more partners recognized that NetBeans increases their productivity by delivering open standards and innovative technology.

What's the list of line items and features that the developers are working on for the next release?

TC: NetBeans 5.5 is our next release. It's all about supporting Java EE 5 – which is a big advance over J2EE 1.4 – but we obviously also want to continue providing a great out-of-box experience for Web development. I don't know if you saw the Visual Web Development demo at JavaOne, but a subset of those features (originally part of Creator) will be in NetBeans 5.5 and we also have a few other things coming down the pike: Subversion support and new features in our GUI Builder will be available through the Update Center.

Where is the NetBeans market growing? Is it in the corporate space, and what geographies are big users?

TC: The NetBeans market is growing in all ways. We've increased our active users by a factor of six since we released NetBeans 4.0 in January 2005. We have over 120 partners who are either building on our platform or using our IDE for development. Our CD Around the World Program has received orders and we've shipped CDs to over 40 different countries. We are huge in Brazil where the community is localizing NetBeans 5.0. We have a large and growing community in China and look for some interesting things coming out of India this coming year.

NetBeans is Open Source. Do most of your contributors still come from the original Xelfi coders in the Czech republic, or you have you grown this to non-Sun employee ?

TC: Most of our developers are currently Sun employees. However, growing beyond this is one of our top priorities for the coming year. With the increased demand and interest in NetBeans, we just cannot continue to do it on our own. The community has some great ideas and developers and it would be silly if we didn't take advantage of that brainpower. In the past six months, we've seen a dramatic increase in translations and plug-in development from the community.

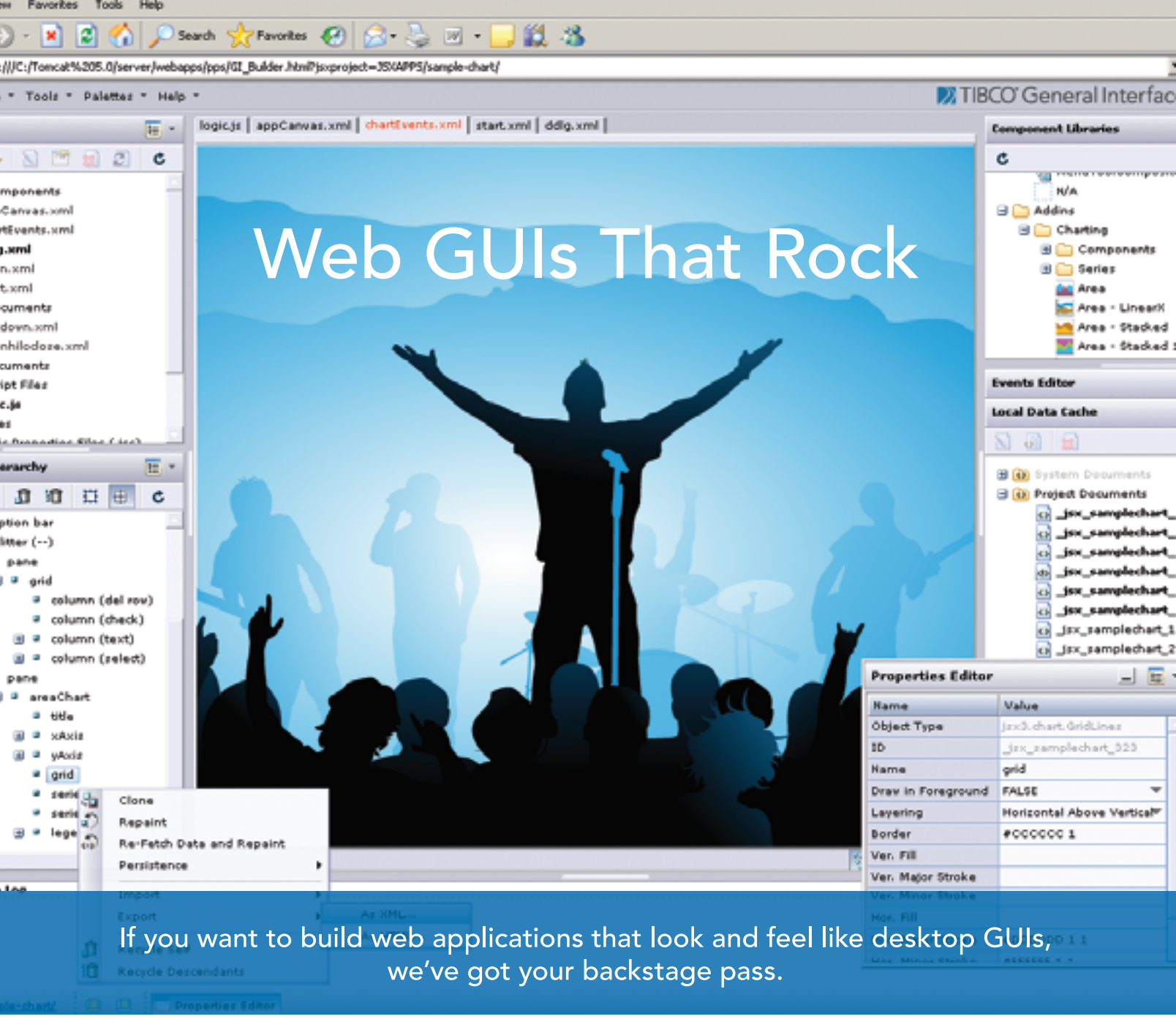
Do NetBeans users tend to be Java EE developers, Java SE, or Java ME?

TC: The thing about NetBeans is that

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

joewinchester@sys-con.com

> "NetBeans 5.0 delivered a lot of new features that blew the Java developer away: the GUI Builder formerly known as Project Matisse, the code-aware collaboration tools, the NetBeans Profiler, and Web framework support"



Why are professional developers choosing TIBCO General Interface?
With more components and tools, TIBCO General Interface™ enables you to create AJAX applications that look and feel like desktop GUIs with astounding speed. No wonder it's the #1 independently rated AJAX Rich Internet Applications toolkit.

SOA and AJAX Rich Internet Applications are changing the way software is developed and deployed. TIBCO helps you make the shift from 3-tier to SOA-based computing to deliver astoundingly rich and agile solutions fast.

TIBCO®
The Power of Now®

Are *you* ready to rock? Learn more at http://developer.tibco.com/.

it enables development end-to-end – from the phone handset to the big back-end server. You can create a Java ME, EE, or SE application with NetBeans. But, I'd have to say that we have more Java SE and EE developers but probably only because the market is larger in those areas. Our support for Java ME is very strong. The NetBeans Mobility Pack has gotten rave reviews and is ahead of the competition in almost every area.

Swing used to, and still does, get beaten up quite a bit over being an emulated widget toolkit. Is this an issue for NetBeans, and are there any plans for SWT tooling?

TC: Not an issue at all. People who are complaining about Swing have probably not used it in a long time. There have been HUGE improvements in JDK 5 (and there are more coming in JDK 6) that boosted performance and increased the ability of Swing applications to "look native" on a number of platforms. Being a 100% Swing application, NetBeans naturally benefits from those. We have no plans for SWT tooling as the value proposition is based on Swing, standards, and 100% Java. SWT is not a request that we hear from our users.

Do Java EE third-party frameworks like Hibernate have a very positive effect? How does NetBeans cope with third-party providers?

TC: The community has responded to the momentum behind NetBeans by creating all sorts of plug-in modules including those for third-party frameworks; we have third party plug-ins from both commercial and Open Source providers. And of course there are some third-party frameworks that are supported in NetBeans out-of-the-box: JUnit, Struts, and Ant.

For Swing how do you see popular frameworks like JGoodies being incorporated into NetBeans?

TC: We are open to anyone. If there's interest from the JGoodies community then we'd like to talk to them. Have them contact me or they can just join our NetBeans community and start contributing.

What are the plans regarding support for the data-binding effort being done by the Swing Labs team and other initiatives going on there being tooled for in NetBeans?

TC: Yes. Data binding is a hot topic in the Java community. As you know there's JSR 295, titled Beans Binding, that's pretty early on. Support for this JSR as well as the Swing Application Framework (JSR 296) is fairly high on the NetBeans priorities list. Some of this work was already shown at NetBeans Day at the JavaOne 2006 conference. You can check out the NetBeans GUI Builder (formerly known as Project Matisse) roadmap here: http://form.netbeans.org/roadmap.html

What are the plans for simplifying working with JTable in NetBeans? Other development languages score high on their ease-of-use development in this area whereas by contrast Java is quite difficult to use.

TC: A key part of any binding solution is the ease with which a developer can create a master detail application. A master detail application typically involves a JTable and a host of components driven by the selection in the JTable. We realize the importance of this scenario, and plan to make it as natural as possible. Developers will be able to drag and drop database tables to create a table, binding the table and detail components to the selection. In addition, configuring an existing JTable will be trivial. Developers will easily be able to configure the headers, number of columns, alignment, formatting, colors; you name it, we'll offer it.

Microsoft understands that good development tools that hide complexity help the adoption of the language such as Visual Basic. Do you see the same relationship between NetBeans and Java?

TC: Absolutely. I like to say that NetBeans allows mere mortals to develop Java EE applications. This is one of our key benefits over other IDEs out there and we do it out-of-the-box. Furthermore, the Java EE 5 specification itself is focused on making thing simpler. But there are still lots of things an IDE can do to help with things like creation of specific classes, use of an entity manager, etc. And NetBeans IDE version 5.5 will provide all that and more.

What is going on in NetBeans in the Java EE space, especially with EJB 3.0 and JSF?

TC: In NetBeans 5.5 we'll have complete support for EJB 3.0 and Java Server Faces 1.2. A developer will be able to create EJB 3.0 entity beans from an existing database automatically. Optionally, a developer can then have the IDE generate a basic JSF application automatically from those entity beans. So think about what this means: without writing any code you can create a basic create/read/update/delete (CRUD) Web application in just a few moments.

Eclipse recently broke away from IBM into its own independent foundation that attracted folks like BEA and Borland and there's talk of Google joining. Does NetBeans have plans for a foundation structure to grow its base of contributing companies?

TC: No. We have seen a huge boost in interest from third-party companies

> "A key part of any binding solution is the ease with which a developer can create a master detail application"

> The NetBeans Enterprise Pack will offer the comprehensive UML modeling capabilities as seen in Sun Java Studio Enterprise"

in NetBeans and they're not at all put off by the lack of a foundation. We have over 120 partners. In some ways it actually gives us an advantage – I suspect that the number of conflicting interests that Eclipse is trying to satisfy is straining its ability to innovate.

There was some hooha in the press recently where Scott McNealy said Oracle was backing NetBeans, and then Larry Ellison promptly denied it. What's going on here?

TC: You really have to ask Larry Ellison that.

Where do you see NetBeans and Java going in the next 10 years?

TC: Wow! That's easy. Java will continue to be the premier development platform across the globe and NetBeans will be the market leader in the tools development space. In the more immediate future, we have begun work on NetBeans 6.0 and we should have a beta out in the spring of 2007. This release will address some of the customer feedback we've gotten about enhancements to the editor. Stay tuned.

I've heard that NetBeans is going to offer design-time and high-level tools soon? What's being provided and when can we see this in NetBeans?

TC: Yes, we've recently announced our plans to make available, very shortly, major elements of Sun Java Studio Enterprise, our premier enterprise-grade development environment for architecting and implementing enterprise applications, as an Open Source project in NetBeans.org. These will be released as the NetBeans Enterprise Pack. In fact, one can already download the preview version of the NetBeans 5.5 Enterprise Pack to evaluate features that includes UML modeling, BPEL-based process orchestration, and XML tooling prior to gaining access to the project in Open Source.

What level of UML support is being provided?

TC: The NetBeans Enterprise Pack will offer the comprehensive UML modeling capabilities as seen in Sun Java Studio Enterprise. Providing support for the UML 2.0 specification, the modeling environment is fully synchronized in real-time with the underlying code and this synchronization is bi-directional in nature. We are specifically addressing the productivity concerns of the enterprise teams over the entire lifecycle going well into the maintenance phase of a project by ensuring that the bi-directional model to code synchronization is achieved without introducing "markers" into the code, allowing for real-time model changes to be affected when the underlying code is changed and vice versa. Of course, ease of use and productivity continues to be our single-minded endeavor and this is also reflected with the rich set of capabilities provided with the diagramming and model navigation capabilities.

You mentioned BPEL orchestration and XML capabilities. What can we expect to see in this space?

TC: With the maturing of the SOA space, it's imperative that the development platform provides teams with the ability to rapidly orchestrate services. The NetBeans IDE already provides for easy creation of Web Services, so it's a natural progression to allow for developers to quickly orchestrate them together. To that end, we've showcased in the NetBeans 5.5. Enterprise Pack preview a BPEL-based orchestration designer to develop the BPEL process orchestration and provided a BPEL runtime to deploy and test.

In developing these Web Services we quickly realized that one of the biggest pain points for an enterprise team is XML document creation and manipulation. If we took into account that a real-world XML schema can potentially be as large as 17K lines of code then it becomes self-evident that a lot of time is spent in creating the schema and getting it right. Therefore it's imperative for a development environment to have the right XML tooling to improve productivity. The NetBeans 5.5 Enterprise Pack preview showcases an XML editor and graphical XML document analysis with built-in queries that allow for easy schema creation and debugging.

So where do you see the NetBeans platform going with all of this?

TC: Productivity as envisaged in a SOA project is no more about just the Java code and an individual developer. Productivity in this day and age is measured in the context of the entire team working and collaborating on creating services configuring and orchestrating them and this scenario introduces a lot of necessary layers of technology, ranging from Web-based and Swing clients to Java EE-based Web Services and BPEL-based orchestrations. The NetBeans platform allows teams to navigate these layers and focus on the business domain at hand. With the modeling and BPEL tooling we're letting the NetBeans community achieve greater levels of team productivity.

Couple this with all of the work already done in ensuring better team collaboration with the version control support and the numerous other innovations brought in via the GUI Builder, NetBeans Mobility Pack for developing mobile applications, NetBeans support for Java EE, and one thing stands out, that NetBeans drives productivity – not just for the developer but for the enterprise team at large.

Tim, thank you for your time.

You're welcome.

AJAX THE EASY WAY

With Java and DWR

by Jon Hoffman

Putting AJAX functionality into your Web application can be a daunting task when you're first learning AJAX. After all you're a Java programmer not a JavaScript programmer. It can also be very frustrating having to learn how the different browsers handle XMLHttpRequests. It's been reported, however, that Internet Explorer 7 will support native XMLHttpRequests rather than requiring the developer to make ActiveX requests. This will make a Web developer's life a lot easier.

For Java Developers there are a number of different frameworks/libraries that hide most of the complexity of developing AJAX-enabled Web applications. For this purposes of this article I'll be using one of those libraries called DWR or Direct Web Remoting (http://getahead.itd.uk/dwr/). I chose DWR because I haven't found another framework/library that's easier to use or as flexible.

DWR is an Open Source Java library, distributed under the Apache License version 2, that lets JavaScript call Java methods as if they were running in the browser, when in fact they're running on the server. DWR isn't tied to any one framework so it should work with any standard servlet container.

There are a number of different ways that you can use DWR, and the extensive documentation shows most of them. I've found that having Java methods return the HTML code that I want displayed is the easiest way to learn DWR. Using it this way also allows for the most flexibility in your AJAX designs.

For this article I'll show you how to use DWR to develop two simple AJAX-enabled Web applications. The first Web application will be an old "Hello World" application. This will show just how easy it is to write an AJAX-enabled Web application with DWR. The second application will display the stats of the players on a baseball team. As you select a position, the names of players that can play that position appear in a dropdown select box. Then you can select the player to see their stats.

You'll need to start by downloading the DWR jar file from http://getahead.ltd.uk/dwr/ and copy it to the lib directory of your Web application. The location of the lib directory will vary depending on what servlet container you're using. DWR is designed to work with any standard servlet container so feel free to use the one you're most comfortable with.

Once the DWR jar file is in the lib directory, you'll need to configure the Web application to use it. Add the following lines to the Web.xml file, located in the WEB-INF directory:

```
<servlet>
    <servlet-name>dwr-invoker</servlet-name>
    <servlet-class>uk.ltd.getahead.dwr.DWRServlet</servlet-class>
    <init-param>
        <param-name>debug</param-name>
        <param-value>true</param-value>
    </init-param>
</servlet>
<servlet-mapping>
    <servlet-name>dwr-invoker</servlet-name>
    <url-pattern>/dwr/*</url-pattern>
</servlet-mapping>
```

The <param-name> and <parm-value> tags between the <init-param> tag is optional but I've found the debug page to be extremely useful, however, you'll want to make sure it's turned off when you deploy the application. The debug page will be discussed in greater detail later in this article.

Now you'll need to create a dwr.xml file in the WEB-INF directory. This is the DWR configuration file and for the "Hello World" application you'll only need to put the following lines in it.

```
<!DOCTYPE dwr PUBLIC
      "-//GetAhead Limited//DTD Direct Web Remoting 1.0//EN"
     "http://www.getahead.ltd.uk/dwr/dwr10.dtd">

<dwr>
  <allow>
    <create creator="new" javascript="ajaxFunctions">
      <param name="class" value="ajaxDemoHello.model.ajaxFunctions"/>
    </create>
  </allow>
</dwr>
```

The DWR configuration file defines what classes DWR can create for remote use by JavaScript. In the example above, I defined "ajaxFunctions" to be used by JavaScript to call a Java class named ajaxFunctions in the ajaxDemoHello.model package. The name of the class doesn't have to be the same

Jon Hoffman is a systems operation manager. His primary responsibility is developing Web-based applications in Java. DWR has allowed Jon to turn Web sites and Web pages into Web applications.

hoffmanj@insanelycrazy.com

name that you defined for JavaScript's use; I just find it convenient to keep the names the same.

There are a couple of restrictions to the names that can be used for classes and methods. You can't use any JavaScript reserve words. Most of the words that are reserved in JavaScript are also reserve words in Java so this is normally not an issue but something to keep in mind.

You'll also want to avoid overloaded methods because it's hit or miss on which method gets called. This is because JavaScript doesn't typecast its variables so DWR has to guess which overloaded method to call based on what the variable contains for data.

As an example, let's say there are two methods that are overloaded, one accepting an int as the argument and the other accepting a String as the argument. You then try to call the method that accepts the String as the argument but the String contains "42." DWR may interpret the 42 as the number 42 and call the method that accepts the int.

The "Hello World" application will have two files, an index.html file that contains the static Web page of the Web application and the ajaxFunctions class that contains the methods that DWR will use to implement the AJAX functionality of the Web page shown in Listing 1.

Once the "Hello World" Web application is deployed, you can test the AJAX functionality by using the DWR's debug page shown in Listing 2. To access the debug page you'll want to point your Web browser to http://{your Web server}:{port}/{your Web app}/dwr/, this will bring up a page that will look like Figure 1.

All classes that are configured in the dwr.xml file will be listed here. To test a class, click on the class name. In this case, there's only one class so click on the ajaxFunctions link and you should see a page that looks like the page in Figure 2.

One of the things that makes DWR so easy is this debug page. Not only does it let you test the AJAX functionality of each method but it also tells you what JavaScript files to include in the Web page to access this class with DWR.

About halfway down the page, you'll see the sayHello() method listed with an Execute button next to it. If you click the button, and everything is configured correctly, you'll see the "Hello from AJAX and DWR" message appear next to the Execute button.

The top of the debug page lists three scripts; two are listed as required and one is listed as optional for the ajaxFunctions class. The examples in this article need a function from the optional utility script, so you'll have to put all three scripts in the index.html page. These scripts should go in the head of the HTML page. You can simply cut each script line from the debug page and paste them into your code.

To display the string returned from the sayHello method, we have to write a JavaScript function to act as the callback function for the sayHello method, but DWR makes this easy too. Listing 3 contains this JavaScript function.

This creates a JavaScript function called displayHello that accepts one argument (remember JavaScript doesn't typecast its variables). The only line in the displayHello function uses the setValue function of the DWR utility script. This function takes the value of the second argument (displaystring) and alters the contents of the element with the id of the first argument (message). This will work with almost all HTML elements that use an id tag.

In the index.html page, change the line that displays "Click me" to the following line:

```
<a onClick='ajaxFunctions.sayHello(displayHello)'>Click Me:</a> <div
id="messages">
```

Clicking on the "Click Me:" will trigger the sayHello method of the ajaxFunctions class. You may remember that the sayHello method doesn't accept an argument, but the line above passes one. The first argument in a method that's called with DWR is the JavaScript callback function that the output of the method is passed to. In this case the output of the sayHello method is passed to the displayHello javaScript function that we created above. The sayHello method doesn't accept any arguments but if it did the arguments would be added after the callback function.

The at the end of the line is where the DWRUtil.setValue will put the hello message. The new index.html file is in Listing 4. If the JavaScript files to include (those listed on the debug page) are different than the ones in Listing 4, change them to match the ones on the debug page.

With this new index.html deployed you'll be able to click on the "Click Me:" and the hello message will appear.

Congratulations, you've created your first AJAX-enabled Web application.

Here's a checklist of what's needed to integrate DWR with a Web application:

I. Copy the dwr.jar file to the lib directory of the Web application.
II. Edit the web.xml file so the Web application will recognize DWR.
III. Create the dwr.xml file to configure DWR.
IV. Write the Java class that DWR will use for the AJAX functionality.
V. Add the AJAX functionality to your Web application.

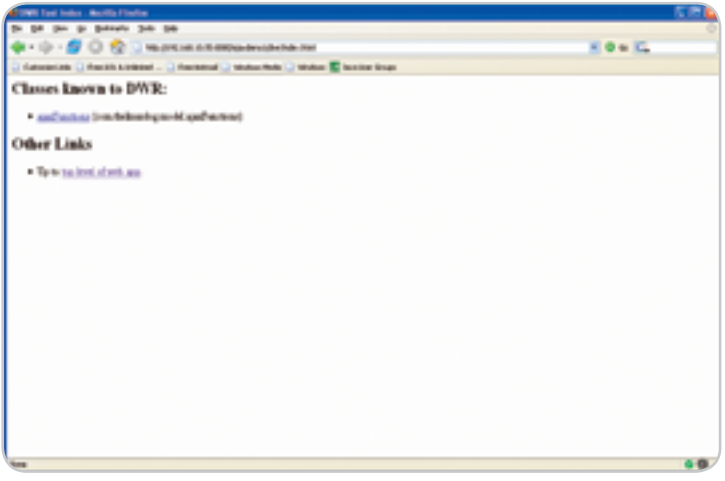


Figure 1

Figure 2

That's really all that's needed to develop an AJAX-enabled Web application with DWR.

Now that you've seen how easy it is to add AJAX functionality to your Web application with DWR, let's create the baseball statistics Web application.

When the application is first started, a list of available positions appears to the left of the screen that can be clicked on. The page should look like Figure 3.

After you click on a position, a list of players that are available for that position will appear in a dropdown selector. Figure 4 shows what it looks like if you selected center fielder.

Once a player is selected, his statistics will appear. Figure 5 shows what this looks like.

Now that we have a basic idea of how the Web application is supposed to function, it's time to start the development. Steps one and two will be the same for almost all applications. Before you go live with the Web application, just remember to disable the debug page.

For the baseball statistics Web application, step three will also be the same as the "Hello World" application. I usually name the Java class that handles the AJAX functionality ajaxFunctions.java just to keep it consistent between applications. Below is the dwr.xml file for the baseball statistics Web application:

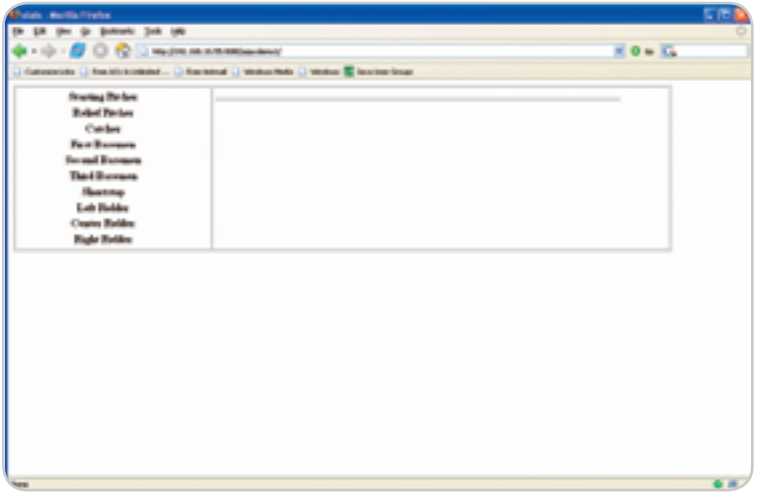


Figure 3

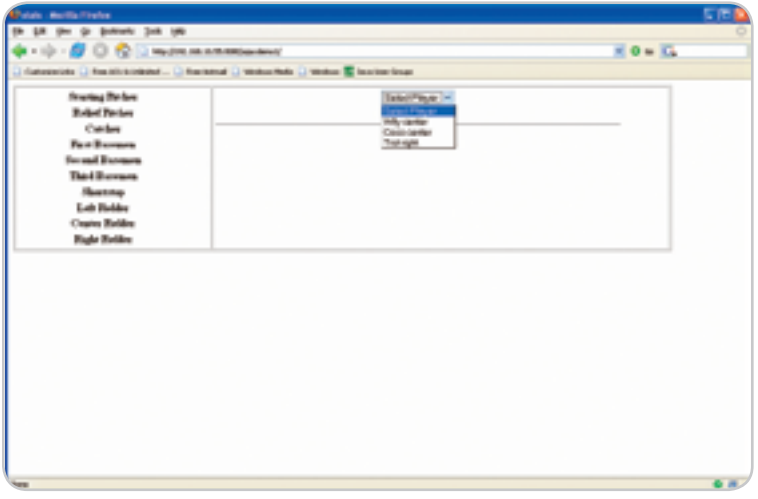


Figure 4

```
<!DOCTYPE dwr PUBLIC
    "-//GetAhead Limited//DTD Direct Web Remoting 1.0//EN"
    "http://www.getahead.ltd.uk/dwr/dwr10.dtd">

<dwr>
  <allow>
    <create creator="new" javascript="ajaxFunctions">
      <param name="class" value="com.thelinuxdog.model.ajaxFunc-
tions"/>
    </create>
  </allow>
</dwr>
```

Step four is to develop the Java class that DWR will use for the AJAX functionality. Before this class can be developed, the class that will handle all the data access functions needs to be written. Normally this class would use a database to store the data, but for simplicity's sake this class will store the data in an array of Strings. Listing 5 contains the dataFunctions class.

The dataFunctions class contains two methods. The first method, returnNames, returns a list of names and the number of the players for a given position. The position is passed to the method through its only argument.

The second method, returnStats, returns an array of Strings that contains the statistics for the player with a given number. The player's number is passed to the method through its only argument.

Listing 6 contains the ajaxFunctions class.

The ajaxFunctions class also contains two methods. The first method, displayNames, displays the dropdown box that contains the players for a given position. The method accepts a String that contains the position that you want to obtain the list of players as its only argument.

The displayNames method begins by getting a list of players who can play the given position from the returnNames method of the dataFunctions. It then begins to build the HTML code for the dropdown box by defining a form with a name of playerform.

The next line defines the select box. The onChange function of the select box calls the displayStats method of the ajaxFunctions class with DWR. This is very similar to the onClick method in the "Hello World" Web application except we're passing two arguments this time because the displayStats method will accept one argument. Keep in mind the first argument passed to a method with DWR is the JavaScript callback function that will accept the output of the Java method. The second argument of the function uses this.options[this.selectedIndex].value to get the number of the selected player and that number is what's passed to the displayStats method.

The options for the select box are then added using the list of names and numbers of the players that were returned from the returnNames method. The value of the option is the player's number and the element is the player's name. The String that represents the HTML code for the select box is then returned, which DWR will then pass to the JavaScript callback function to display on the Web page.

BACKBASE

AJAX for Java

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

The Backbase AJAX Java Edition:

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

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

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

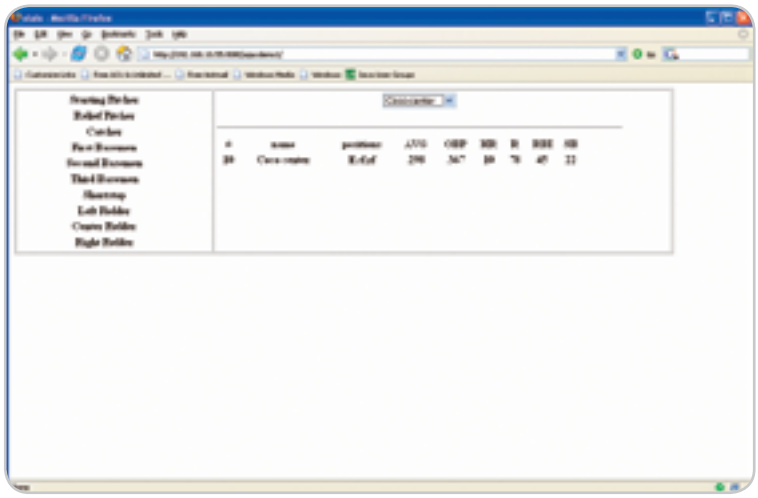


Figure 5

The second method in the ajaxFunctions class is called displayStats. This method returns the HTML code to display the statistics for the player with the number that's passed to the displayStats method in its only argument. This function is similar to the returnNames method. It begins by getting the data needed from the returnsStats method of the dataFunctions class and then builds the HTML code to display the statistics. The method then returns the String that contains the HTML code to display the statistics.

The Web application can be deployed at this point to test the ajaxFunctions class with the DWR debug page. After the application is deployed, point the Web browser to http://{your Web server}:{port}/{your Web app}/dwr/ and you should see the debug page if everything is configured correctly. If you click on the ajaxFunctions link, you'll be taken to the ajaxFunctions debug page, which will also list the JavaScript files needed by the Web application to use the ajaxFunctions class with DWR. Listing 7 contains the index.jsp file for the baseball statistics Web application. Remember to substitute the links to the JavaScript files with the ones from your ajaxFunctions debug page.

You may have noticed that in the ajaxFunctions class the displayStats method was called by the displayNames method. Calling an AJAX method from another AJAX method is perfectly fine under one condition, the JavaScript functions needed by the AJAX method has to be on the static HTML page and not created by the AJAX method. For example, when the displayStats method is called it sends the output to the JavaScript callback function called displayStatsJS. Therefore, the displayStatsJS function needs to be in the static HTML page, or in this case the static jsp page called index.jsp.

That is all there is to developing an AJAX-enabled Web application with DWR. The next question is "what do you do if the AJAX functionality doesn't function properly?" The debug page created by DWR is the first tool that I use for debugging my AJAX-enabled applications. It's good for testing if DWR is configured properly and if the functions called through DWR are working properly.

If everything appears to work properly through the debug page but the AJAX part of your Web application still isn't working, you'll need to know a little bit of JavaScript to debug your application. The most useful JavaScript command for debugging is the alert command that pops up an alert window. For example, the command alert("Hello"); pops up an alert box with the word "Hello" in it. This can be useful to see if you're getting to your callback functions and to test what's in the variables.

If I were to change the displayNameJS javaScript function to:

```
function displayNamesJS(data)
            {
      alert(data);
            DWRUtil.setValue("names",data);
            }
```

everytime the displayNamesJS function is called, an alert pop-up box that contains the HTML code that's going to be displayed on the page would appear

As you can see it's pretty easy to develop AJAX-enabled Web applications with DWR. DWR handles all of the compatibility issues with the different browsers and the asynchronous calls to the server so all you have to do is write the Java classes that DWR will use.

I've been using DWR version 1.x for a while now in my development environment and found it to be very stable. They are in the process of developing a version 2 that will introduce "Reverse Ajax," which will allow Java on the server to send JavaScript to the client.

In this article I showed how to develop AJAX-enabled Web applications by having DWR pass HTML code back to the browser. To me this is the easiest and most flexible way of using DWR, but not the only way. The DWR Web site has very good documentation and examples (http://getahead.itd.uk/dwr/). I'd really recommend that if you plan on using DWR in your production environment that you spend a little time reading the full documentation to find the best way of using DWR for your project.

Listing 1: ajaxFunctions class

```
package ajaxDemoHello.model;

public class ajaxFunctions {

    public ajaxFunctions() {
    }

    public String sayHello() {
        String returnString="";
        returnString += "Hello from AJAX and DWR";
        return returnString;
    }
}
```

Listing 2: index.html file without any AJAX functionality.

```
<html>

    <head>

        <title>HelloWorld</title>

     </head>

     <body>

                  Click Me

     </body>

</html>
```

–continued on page 32

Experience for yourself

...why Yakov says Adobe Flex 2 is "Very potent".

Adobe thanks Java Developer's Journal and Yakov Fain for selecting Adobe Flex 2 for the Editors' Choice award. **Better By Adobe.**™

www.adobe.com/go/try_jdjchoice

Adobe

Listing 3

```
<script type='text/javascript'>
      function displayHello(displaystring)
      {
            DWRUtil.setValue("message",displaystring);
      }
</script>
```

Listing 4

```
<html>
  <head>
    <title>HelloWorld</title>
  <script type='text/javascript' src='/ajaxDemo/dwr/interface/ajax-
Functions.js'></script>
  <script type='text/javascript' src='/ajaxDemo/dwr/engine.js'></
script>
  <script type='text/javascript' src='/ajaxDemo/dwr/util.js'></script>
  <script type='text/javascript'>
    function displayHello(displaystring)
    {
         DWRUtil.setValue("message",displaystring);
    }
  </script>

  </head>
  <body>
        <a onClick='ajaxFunctions.sayHello(displayHello)'>Click Me:</
a> <div id="message">
  </body>
</html>
```

Listing 5: dataFunctions.java

```
package com.thelinuxdog.model;

import java.util.ArrayList;
import java.util.List;

/**
 *
 * @author Jon Hoffman
 */
public class dataFunctions {
    String[][] data = {
        {"0","Jason Catcher", "c", ".307", ".345", "10", "87", "76",
"0"},
        {"1", "Doug Catcher", "c", ".202", ".290", "1", "15", "25",
"0"},
        {"2", "Kevin infielder", "1b,3b", ".320", ".410", "2", "95",
"76", "2"},
        {"3", "David infielder", "1b" ,".302", ".378", "43", "140",
"102", "0"},
        {"4", "Mark second", "2b", ".298", ".315", "10", "79", "89",
"5"},
        {"5", "Alex second", "2b,ss", ".250", ".290", "2", "10",
"22", "4"},
        {"6", "Alex short", "ss", ".210", ".250", "9", "45", "50",
"2"},
        {"7", "Mike Third", "3b", ".289", ".330", "25", "70", "80",
"0"},
        {"8", "Manny Left", "lf,rf", ".325", ".414", "42", "143",
"101","1"},
        {"9", "Wily center", "lf,cf,rf", ".287", ".335", "25", "67",
"56", "5"},
        {"10", "Coco center", "lf,cf,rf", ".298", ".367", "10","78",
"45","22"},
        {"11", "Trot right", "cf,rf", ".289", ".330","15", "76",
"67", "0"},
        {"12", "Curt Ace", "sp", "3.14", "1.21","21", "5",
"0","212"},
        {"13", "Josh Ace", "sp", "3.25", "1.14", "20", "6",
"0","189"},
        {"14", "Tim Starter", "sp", "4.19","1.45", "15", "7", "0",
"140"},
        {"15", "Johnaton SuperCloser", "rp", "0.89", "0.50", "4",
"0", "55", "110"}

    };

    /** Creates a new instance of dataFunctions */
    public dataFunctions() {
    }

    public List returnNames(String position)
    {
        List returnList = new ArrayList();
        for (int i=0; i<data.length; i++)
        {
            String[] pos = data[i][2].split(",");
            for (int j=0; j<pos.length; j++)
            {
                if (pos[j].equalsIgnoreCase(position))
                {
                    returnList.add(data[i][0]);
                    returnList.add(data[i][1]);
                }
            }
        }
        return returnList;
    }

    public String[] returnStats(String number)
    {
        String[] returnString = null;
        for (int i=0; i<data.length; i++)
        {
             if (data[i][0].equalsIgnoreCase(number))
                    returnString = data[i];

        }
        return returnString;
    }

}
```

Listing 6: ajaxFunctions.java

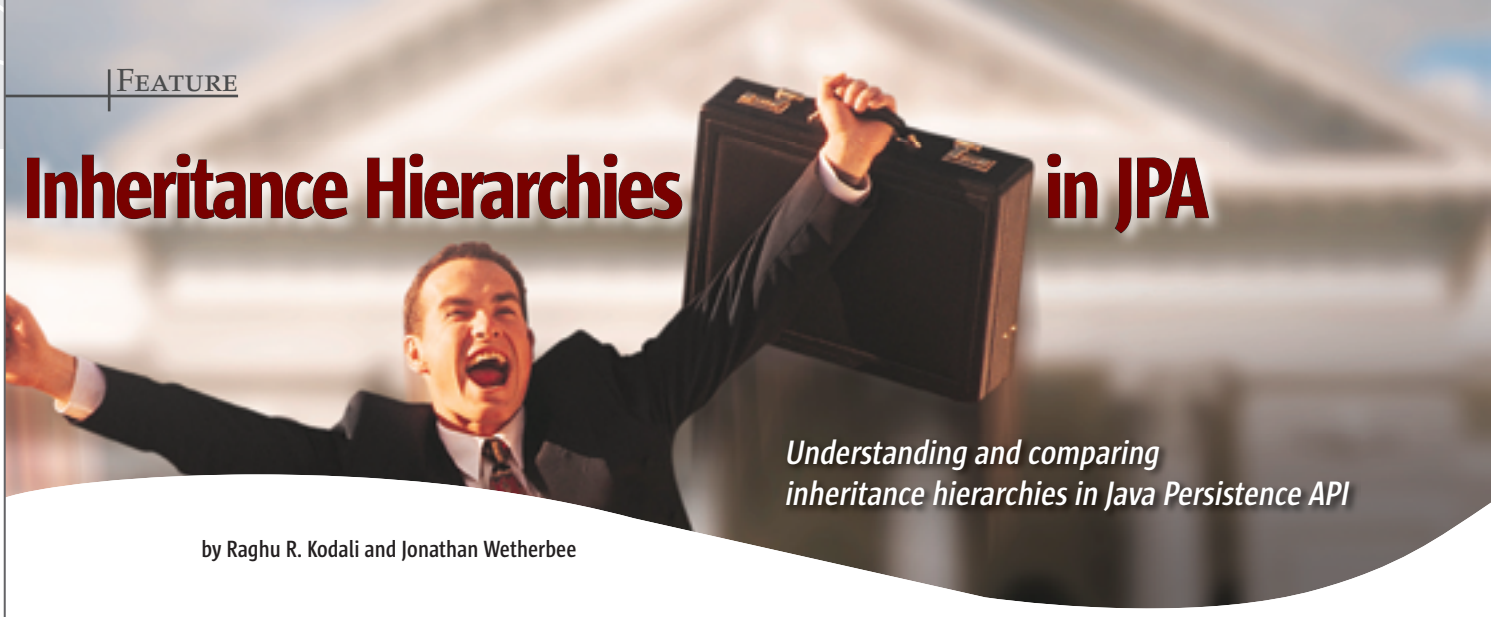
```
package com.thelinuxdog.model;

import java.util.List;
```

```
public class ajaxFunctions {
     public ajaxFunctions() {
    }

     public String displayNames(String pos) {
        String returnString = "";
        dataFunctions df = new dataFunctions();
        List names = df.returnNames(pos);
        returnString += "<form name=playerform>";
        returnString += "<select name=player onChange='ajaxFunctions.
displayStats(displayStatsJS,this.options[this.selectedIndex].value)'>";
        returnString += "<option value=1000>Select Player</option>";
        for (int i=0; i<(names.size()/2); i++)
        {
            int key=i*2;
            returnString += "<option value="+ names.get(key) +">" +
names.get(key+1) + "</option>";
        }
        returnString += "</select>";
        return returnString;
    }

    public String displayStats(String num) {

        String returnString = "";
        if (!num.equalsIgnoreCase("1000"))
        {
            String fieldsPitchers[] = {"#","name","positions", "ERA",
"WHIP", "W","L","Saves","K"};
            String fieldsHitters[] = {"#","name","positions","AVG","O
BP","HR","R","RBI","SB"};
            dataFunctions df = new dataFunctions();
            String[] stats = df.returnStats(num);
            returnString += "<table width=90% border=0><tr>";

            for (int i=0; i<9; i++)
            {
                if(stats[2].startsWith("sp") || stats[2].
startsWith("rp"))
                    returnString += "<th>" + fieldsPitchers[i] +
"</th>";
                else
                    returnString += "<th>" + fieldsHitters[i] + "</
th>";;
            }
            returnString += "</tr><tr>";
            for (int i=0; i<stats.length; i++)
                returnString += "<th>" + stats[i] + "</th>";
            returnString += "</tr></table>";
        }
        return returnString;
    }
}
```

Listing 7: index.jsp

```
<html>
    <head>
        <title>stats</title>
        <script type='text/javascript' src='/ajaxdemo1/dwr/interface/
ajaxFunctions.js'></script>
        <script type='text/javascript' src='/ajaxdemo1/dwr/engine.
js'></script>
        <script type='text/javascript' src='/ajaxdemo1/dwr/util.js'></
script>
        <script type='text/javascript'>

            function displayNamesJS(data)
            {
                DWRUtil.setValue("names",data);
            }

            function displayStatsJS(data)
            {
                DWRUtil.setValue("stats",data);
            }
        </script>
    </head>
    <body>
        <table width=90% border=1>
            <tr>
            <th width=30% valign=top>
                <table width=90% border=0>
                    <tr><th><a onClick='ajaxFunctions.displayNames
(displayNamesJS,"sp")'>Starting Pitcher</a></th></tr>
                    <tr><th><a onClick='ajaxFunctions.displayNames
(displayNamesJS,"rp")'>Relief Pitcher</a></th></tr>
                    <tr><th><a onClick='ajaxFunctions.displayNames
(displayNamesJS,"c")'>Catcher</a></th></tr>
                    <tr><th><a onClick='ajaxFunctions.displayNames
(displayNamesJS,"1b")'>First Basemen</a></th></tr>
                    <tr><th><a onClick='ajaxFunctions.displayNames
(displayNamesJS,"2b")'>Second Basemen</a></th></tr>
                    <tr><th><a onClick='ajaxFunctions.displayNames
(displayNamesJS,"3b")'>Third Basemen</a></th></tr>
                    <tr><th><a onClick='ajaxFunctions.displayNames
(displayNamesJS,"ss")'>Shortstop</a></th></tr>
                    <tr><th><a onClick='ajaxFunctions.displayNames
(displayNamesJS,"lf")'>Left Fielder</a></th></tr>
                    <tr><th><a onClick='ajaxFunctions.displayNames
(displayNamesJS,"cf")'>Center Fielder</a></th></tr>
                    <tr><th><a onClick='ajaxFunctions.displayNames
(displayNamesJS,"rf")'>Right Fielder</a></th></tr>
                </table>
            </th>
            <th width=70% valign=top>
                <table width=90% border=0>
                    <tr><th><div id=names /></th></tr>
                    <tr><th><hr width=100%></th></tr>
                    <tr><th><div id=stats /></th></tr>
                </table>
            </th>

        </table>


    </body>
</html>
```

Inheritance Hierarchies in JPA



Understanding and comparing inheritance hierarchies in Java Persistence API

by Raghu R. Kodali and Jonathan Wetherbee

The persistence model introduced in EJB 3.0 as a replacement for entity beans is known as the Java Persistence API (JPA). The JPA borrows from both proprietary and open source models, such as Oracle TopLink, Hibernate, Spring, and other frameworks, which have gained traction as popular alternatives to the often heavyweight and cumbersome persistence directives required by earlier EJB revisions. Among the new features introduced in EJB 3.0 through the JPA is support for entity inheritance. In this article, we will examine inheritance strategies supported by the JPA and apply these strategies to a simple entity hierarchy, exploring the strengths and weaknesses of each approach. This comparison is intended to help you understand how to set up entity hierarchies, and to decide which mapping approach to take for the entity hierarchies in your own application.

Raghu R. Kodali is a consulting product manager and SOA evangelist for Oracle Application Server. He leads next-generation SOA initiatives and J2EE feature sets for Oracle Application Server, with particular expertise in EJB, J2EE deployment, Web services, and BPEL. He holds a Masters degree in Computer Science and is a frequent speaker at technology conferences. Raghu is also a technical committee member for the OASIS SOA Blueprints specification, and a board member of Web Services SIG in OAUG. He maintains an active blog at Loosely Coupled Corner (www.jroller.com/page/raghukodali).

raghu.kodali@oracle.com

Mapping JPA Entity Inheritance Hierarchies

Java has supported single-class inheritance — in which a non-interface class can extend a single other class — since its inception. While it's been common practice to exploit the code reuse and polymorphism benefits of inheritance in many areas of the business domain, data inheritance hasn't been well handled in the EJB persistence domain until now. This has been a major shortcoming, since in the real world, data is often hierarchical, and the lack of standard, built-in support for the inheritance of data objects has required countless workarounds and headaches. Leveraging the ease of use of Java SE annotations, JPA delivers declarative support for defining and mapping entity inheritance hierarchies, including abstract entities and polymorphic relationships and queries.

JPA Entity Inheritance Mapping Strategies

JPA provides declarative support for three main implementation strategies that dictate how the entities in a hierarchy map to underlying tables. To illustrate how these three strategies are manifested in code, Figure 1 shows a sample entity hierarchy that demonstrates both inheritance and polymorphic relationships.

In Figure 1, the **Person** entity serves as the root class in an entity hierarchy, and is extended by the **Employee** entity. **Employee** is further specialized to produce two other entities: **FullTimeEmployee** and **PartTimeEmployee**.

Object/Relational Inheritance Mapping Strategies

Now that we've defined our entity hierarchy, let's look at how each of the three O/R strategies supported natively by JPA can be used to map this **Person** entity hierarchy, and the associated **Address** entity, to a relational schema. Here's a summary of each strategy defined by the **InheritanceType** enum:

```
public enum InheritanceType
{SINGLE_TABLE, JOINED, TABLE_PER_CLASS};
```

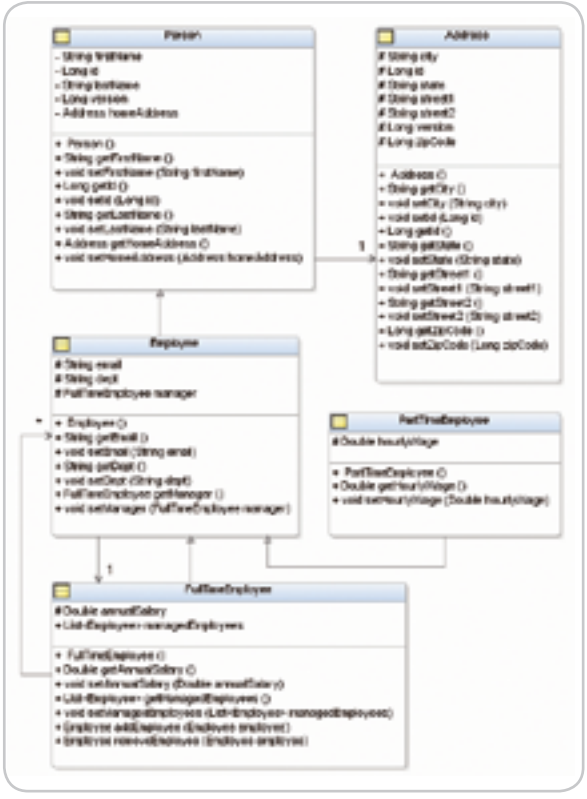


Figure 1 An entity type hierarchy, rooted in the base entity Person, showing relationships to entities both inside and outside the hierarchy

- **SINGLE_TABLE:** Single-table-per-class inheritance hierarchy. This is the default strategy. The entity hierarchy is essentially flattened into the sum of its fields, and these fields are mapped down to a single table.
- **JOINED:** Common base table, with joined subclass tables. In this approach, each entity in the hierarchy maps to its own dedicated table that maps only the fields declared on that entity. The root entity in the hierarchy is known as the base table, and the tables for all other entities in the hierarchy join with the base table.
- **TABLE_PER_CLASS:** Single-table-per-outermost concrete entity class. This strategy maps each leaf (i.e., outermost, concrete) entity to its own dedicated table. Each such leaf entity branch is flattened, combining its declared fields with the declared fields on all of its super-entities, and the sum of these fields is mapped onto its table.

Single-Table-per-Class Inheritance Hierarchy

The default inheritance mapping strategy is SINGLE_TABLE in which all the entities in the class hierarchy map onto a single table. A dedicated discriminator column on this table identifies the specific entity type associated with each row, and each entity in the hierarchy is given a unique value to store in this column. By default, the discriminator value for an entity is its entity name, although an entity may override this value using the **@DiscriminatorValue** annotation. This approach performs well for querying, since only a single table is involved, and if your type hierarchy can abide by the practical limitations, this is probably the best approach to use. Figure 2 shows a diagram of a schema that maps our example entities using the SINGLE_TABLE strategy.

All the properties across the entity hierarchy rooted by the **Person** entity map to columns on a single table, **CH04_ST_PERSON**. This table holds a foreign key reference, bound to the **HOME_ADDRESS** column, to **CH04_ST_ADDRESS**, which is mapped to the **Address** entity. This column also has a unique key constraint, ensuring that each row in the **CH04_ST_ADDRESS** table corresponds to at most one row in the **CH04_ST_PERSON** table. It also holds a foreign key reference, using the **MANAGER** column, back to itself. This foreign key isn't constrained to be unique, indicating that multiple rows may hold the same value in their **MANAGER** column.

Listings 1 through 4 show how the entities in the **Person** hierarchy are mapped using the **SINGLE_TABLE** inheritance strategy. The strategy is declared on the root entity in the hierarchy and applies to all sub-entities in the hierarchy as well.

Let's take a look at the annotations that were introduced.

The @DiscriminatorColumn Annotation

By default, the persistence manager looks for a column named **DTYPE** in the root entity's table (**CH04_ST_PERSON**, in this case). In our example, we've named the discriminator column **TYPE**, so we explicitly annotate this setting, using the **@DiscriminatorColumn(name =**

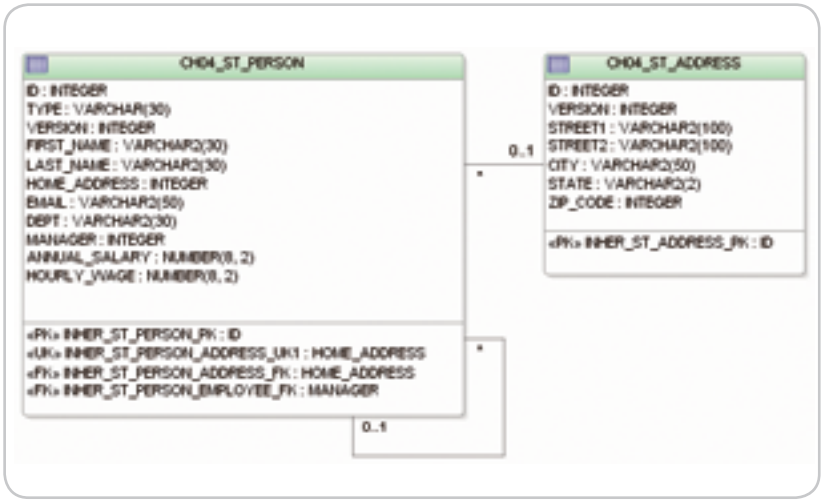


Figure 2 The CH04_ST_PERSON table holds all entity instances in the entity hierarchy rooted by Person. The CH04_ST_ADDRESS table holds the associated Address instances

Figure 3 A schema that maps our example entities using the JOINED strategy. Each entity in the hierarchy has its own table to persist its declared fields. The table CH04_JOIN_ADDRESS holds associated Address instances

"TYPE") annotation. Were we to use a column named **DTYPE**, we could have skipped this annotation altogether and accepted the default value.

The @DiscriminatorValue Annotation

Each concrete entity declares, either explicitly or by tacitly accepting the default, a unique discriminator value that serves to identify the concrete entity type associated with each row in the table. The discriminator value defaults to the entity name, and in this example we've accepted this default value for each of the entities in the hierarchy.

Considerations	Pros and Cons
Design-time considerations	This mapping approach works well when the type hierarchy is fairly simple and stable. Adding a new type to the hierarchy and adding fields to existing supertypes simply involves adding new columns to the table; though in particularly large deployments, this may have an adverse impact on the index and column layout inside the database. If your hierarchy may outgrow the column limitations of a single table, which is typically 256 columns, or if for some reason you need to map more than one very large field to inline LOB (Locator Object) columns, you may have to introduce an @SecondaryTable mapping. In this case, it might be wiser to adopt one of the approaches that follow.
Performance impact	This strategy is very efficient for querying across all types in the hierarchy, or specific types. No table joins are required by the internal persistence framework—only a WHERE clause listing the type identifiers. In particular, relationships involving types that employ this mapping strategy are very performant.

Table 1 Pros and Cons of a SINGLE_TABLE Inheritance Strategy

Pros and Cons of the SINGLE_TABLE Strategy

Table 1 offers a look at some of the strengths and weaknesses of the SINGLE_TABLE entity inheritance strategy.

Common Base Table with Joined Subclass Tables

In the JOINED strategy, each entity in the hierarchy introduces its own table but only to map fields that are declared on that entity type. The root entity in the hierarchy maps to a root table that defines the primary key structure to be used by all tables in the entity hierarchy, as well as the discriminator column and optionally a version column. Each of the other tables in the hierarchy defines a primary key that matches the root table's primary key, and optionally adds a foreign key constraint from their ID column(s) to the root table's ID column(s). The non-root tables don't hold a discriminator type or version columns. Since each entity instance in the hierarchy is represented by a virtual row that spans its own table as well as the tables for all of its super-entities, it eventually joins with a row in the root table that captures this discriminator type and version information. Querying all the fields of any type requires a join across all the tables in the supertype hierarchy.

Figure 3 illustrates the schema that maps our entities using the JOINED inheritance strategy. As in the previous example, we've prefixed the tables with the strategy indicator, in this case **CH04_JOIN_.**

Listings 5 shows how the abstract entity in the **Person** hierarchy is mapped using the **JOINED** inheritance strategy.

Pros and Cons of the JOINED Strategy

Table 2 offers a look at some of the strengths and weaknesses of a JOINED entity inheritance strategy.

Single-Table-per-Outermost Concrete Entity Class

This inheritance mapping option maps each outermost concrete entity to its own dedicated table. Each table maps all of the fields in that entity's entire type hierarchy; since there's no shared table, no columns are shared. The only table structure requirement is that all tables must share a common primary key structure, meaning that the name(s) and type(s) of the primary key column(s) must match across all tables in the hierarchy. For good measure, Figure 4 illustrates our third type hierarchy using the joined table approach, which demonstrates the use of the single-table-per-outermost entity subclass strategy. The tables are required to share nothing in common except the structure of their primary key, and since the table implicitly identifies the entity type, no discriminator column is required. With this inheritance mapping strategy, only concrete entities — **FullTimeEmployee**, **PartTimeEmployee**, and **Address** in our example — require tables.

Listing 6 shows how the Person entity is mapped and annotated.

Pros and Cons of the TABLE_PER_CLASS Strategy

Table 3 offers a look at some of the strengths and weaknesses of the TABLE_PER_CLASS entity inheritance strategy.

Comparison of O/R Implementation Approaches

Now that we've explored the three inheritance mapping implementations, let's look at some of the characteristics of a class inheritance hierarchy to consider when choosing which implementation approach to use for your type hierarchies. The following list contains subjective questions about your own entity hierarchies. They don't have precise answers, but are meant to stimulate design consideration when building your application.

- Class hierarchies can be static, with a fixed number of sub-types, or they can be dynamic, with varying numbers of sub-types. How often will you need to incorporate new sub-types into your hierarchy?
- Hierarchies can be deep, with lots of sub-classes, or they can be shallow, with only a few. How granular is your hierarchy?
- The types in a hierarchy may diverge greatly, with very different sets of properties on the sub-classes than the base class, or with very little difference in properties.
- How much do the persistent property sets of your entities diverge from one another? Will other entities define relationships with classes in this type hierarchy, and if so, will the base classes frequently be the referenced type?
- Will types in this hierarchy be frequently queried, updated, or deleted? How will the presence or absence of SQL JOIN or UNION operations impact your application's performance?
- During the life of your application, how frequently will you be updating the structure of the type hierarchy itself? The impact of this type of change varies for each inheritance strategy, with considerations that include the following:
  - Adding or removing new types to the hierarchy (as when refactoring classes).
  - Adding, removing, or modifying fields on an entity in the hierarchy.
  - Adding, removing, or modifying relationships involving types in this hierarchy.

Summary

To support entity inheritance hierarchies, the JPA offers three mapping approaches for developers to choose from:

- **SINGLE_TABLE:** Single-table-per-class inheritance hierarchy
- **JOINED:** Common base table, with joined subclass tables
- **TABLE_PER_CLASS:** Single-table-per-outermost concrete entity class

Jonathan Wetherbee is a consulting engineer and tech lead for EJB development tools on Oracle's JDeveloper IDE. He has over 10 years of experience in development at Oracle, working on a variety of O/R mapping tools with responsibility for Oracle's core EJB toolset since EJB 1.1. Before joining Oracle's development staff, Jonathan was a product manager for Oracle's CASE (Computer Aided Software Engineering) tools. In 1999 he received a patent for his work on integrating relational databases in an object-oriented environment. Jonathan received a bachelor of science in cognitive science from Brown University.

jon.wetherbee@oracle.com

OpenLaszlo

Advancing the Web Experience™

Hit the road to freedom with OpenLaszlo, the only open source advanced Ajax development platform that lets you choose between Flash® and DHTML for running your applications.

Open Source:
Create and control cost-effective, mission-critical web applications

Open Standards:
Build web applications with the most mature and advanced AJAX (JavaScript and XML) development platform

Open Architecture:
Designed to support multiple runtimes, including Flash, DHTML and embedded devices (mobile and TVs)

Laszlo®

Download today at www.openlaszlo.org

Considerations	Pros and Cons
Design-time considerations	Introducing a new type to the hierarchy, at any level, simply involves interjecting a new table into the schema. Subtypes of that type will automatically join with that new type at run time. Similarly, modifying any entity type in the hierarchy by adding, modifying, or removing fields affects only the immediate table mapped to that type. This option provides the greatest flexibility at design time, since changes to any type are always limited to that type's dedicated table.
Performance impact	This approach does not suffer from the use of UNION operations, but inherently requires multiple JOIN operations to perform just about any query. Querying across all instances initially involves only a single query of the top most base entity's table to retrieve a list of all primary keys of instances in the hierarchy. Due to the presence of the discriminator column in the base entity's table, resolution of these instances into entity classes can be efficient, depending on the lazy loading strategies employed by the persistence manager implementation.

Table 2 Pros and Cons of the JOINED Inheritance Strategy

Considerations	Pros and Cons
Design-time considerations	As new outermost concrete types are introduced into the hierarchy, new tables are added. This is nice because no existing tables (nor their data) need be aware. However, since each type maps all of its supertype fields as well, introducing a new field on a base class, or a new base entity itself, requires modifying the tables for all affected subtypes across the hierarchy to map any newly introduced fields.
Performance impact	Querying across multiple types requires a UNION select statement, which is not very performant, but querying a single type is very efficient, since only one table is involved in the query. Relationships involving supertypes in this hierarchy should be avoided, since they will necessarily require this UNION operation to resolve to concrete subtype instances.

Table 3 Pros and Cons of a TABLE_PER_CLASS Inheritance Strategy

Figure 4 A schema that maps our example entities using the TABLE_PER_CLASS strategy. Concrete leaf entities are mapped to dedicated tables that contain columns that map all of their declared and inherited fields

In this article we took a sample entity inheritance hierarchy and explored how to map the entities in this hierarchy using each of these three inheritance mapping strategies. We also examined some of the strengths and weaknesses in each strategy, to help you choose the approach that best serves each of the entity hierarchies in your application.

Now that you are familiar with how to set up an entity inheritance hierarchy in the JPA, you may wish to explore the many related areas also introduced in the JPA, including polymorphic relationships and JPQL queries, the use of non-entity mapped superclasses, and composite primary keys. For an examination of these features, with code samples, check out *Beginning EJB 3 Application Development: From Novice to Professional* (Apress, 2006).

Listing 1: Person.java, an Abstract Root Entity in a SINGLE_TABLE Inheritance Hierarchy

```
/*
* Person: An abstract entity, and the root of a SINGLE_
TABLE hierarchy
*/
@Entity
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name = "TYPE")
@NamedQuery(name = "Person.findAll", query = "select o
from Person o")
@SequenceGenerator(name = "PersonIdGenerator",
sequenceName = "CH04_ST_PERSON_SEQ", initialValue = 100,
allocationSize = 20)
@Table(name = "CH04_ST_PERSON")
public abstract class Person
implements Serializable
{
@Column(name = "FIRST_NAME")
private String firstName;
@Id
@Column(nullable = false)
@GeneratedValue(generator="PersonIdGenerator")
private Long id;
@Column(name = "LAST_NAME")
private String lastName;
@Version
private Long version;
@OneToOne(cascade = { CascadeType.ALL })
@JoinColumn(name = "HOME_ADDRESS", referencedColumnName
= "ID")
private Address homeAddress;
public Person() {
}
```

```
/* get/set methods... */
}
```

Listing 2: Employee.java, an Abstract Intermediate Entity in a SINGLE_TABLE Inheritance Hierarchy

```
/*
* Employee: An abstract entity
*/
@Entity
@NamedQuery(name = "Employee.findAll", query = "select o from
Employee o")
public abstract class Employee
extends Person
implements Serializable
{
protected String email;
protected String dept;
@ManyToOne(cascade = { CascadeType.ALL })
@JoinColumn(name = "MANAGER", referencedColumnName = "ID")
protected FullTimeEmployee manager;
public Employee() {
}
/* get/set methods... */
}
```

Listing 3: FullTimeEmployee.java, a Concrete Leaf Entity in a SINGLE_TABLE Inheritance Hierarchy

```
/*
* FullTimeEmployee: A concrete leaf entity */
@Entity
@NamedQuery(name = "FullTimeEmployee.findAll",
query = "select o from FullTimeEmployee o")
@DiscriminatorValue("FullTimeEmployee") // Illustrating the default
value
public class FullTimeEmployee
extends Employee
implements Serializable
{

@Column(name = "ANNUAL_SALARY")
protected Double annualSalary;
@OneToMany(mappedBy = "manager", cascade = { CascadeType.ALL })
public List<Employee> managedEmployees;

public FullTimeEmployee() {
}
/* get/set methods... */
}
```

Listing 4: PartTimeEmployee.java, a Concrete Leaf Entity in a SINGLE_TABLE Inheritance Hierarchy

```
/*
* PartTimeEmployee: A concrete leaf entity
*/
@Entity
@NamedQuery(name = "PartTimeEmployee.findAll",
query = "select o from PartTimeEmployee o")
public class PartTimeEmployee
extends Employee
implements Serializable
{
@Column(name = "HOURLY_WAGE")
protected Double hourlyWage;
public PartTimeEmployee() {
}
/* get/set methods... */
}
```

Listing 5: Person.java, an Abstract Root Entity in a JOINED Inheritance Hierarchy

```
/*
* Person: An abstract entity, and the root of a JOINED hierarchy
*/
@Entity
@Inheritance(strategy = InheritanceType.JOINED)
@DiscriminatorColumn(name = "TYPE")
@NamedQuery(name = "Person.findAll", query = "select o from Person
o")
@SequenceGenerator(name = "PersonIdGenerator",
sequenceName = "CH04_JOIN_PERSON_SEQ", initialValue = 100,
allocationSize = 20)
@Table(name = "CH04_JOIN_PERSON")
public abstract class Person
implements Serializable
{
/* The class body is identical across all inheritance strategies
*/
}
```

Listing 6: Person.java, an Abstract Root Entity in a TABLE_PER_CLASS Inheritance Hierarchy

```
/*
* Person: An abstract entity, and the root of a TABLE_PER_CLASS
hierarchy
*/
@Entity
@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
@DiscriminatorColumn(name = "TYPE")
@NamedQuery(name = "Person.findAll", query = "select o from Person
o")
@SequenceGenerator(name = "PersonIdGenerator",
sequenceName = "CH04_TPC_PERSON_SEQ", initialValue = 100,
allocationSize = 20)
@Table(name = "CH04_TPC_PERSON")
public abstract class Person
implements Serializable
{
/* The class body is identical across all inheritance strategies
*/
}
```

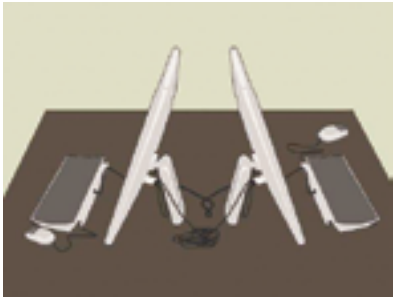
Fault Tolerance with Open Source and JVM Clustering

A marriage made in Java

by Ari Zilka

Many in the Open Source community (including the camps following Tomcat, Geronimo, Struts, Spring, and Hibernate) have chosen to focus on solving problems of developer efficiency and software elegance, and are sometimes forced to leave production operating characteristics such as HA (high availability)/fault tolerance and central management control for future releases. Or, in some cases, the elegance of the framework stems from its light-weight nature and thus the user community as a whole can't be made to suffer the complexities of clustering and HA for the needs of the few.

Combine this future feature roll-out with the current exponentially increasing adoption of Open Source in enterprise applications – and the result is that enterprises are running some applications with little to no fault tolerance. This leaves the enterprise IT manager stuck between a rock and a hard place, because on the one hand enterprises demand HA at reasonable levels of efficiency, but on the other Open Source applications are cheap to acquire, easy to develop, (albeit sometimes hard to operate), and put out the fires immediately without having to mess with budget approvals. Hands downs, Open Source applications



are deployed more frequently than any other software stack, and the IT manager can either ignore the lack of fault tolerance and plan to deal with issues if and when they arise or try to inject HA via additional customization. Solving HA is tricky business as IT managers tend to not want to edit their Open Source framework internals as a one-off.

Therefore, in response to this growing issue, more than five Open Source clustering projects have recently emerged, but the task of integrating them into development frameworks is pretty significant and will realistically take quite some time. Java serialization (some architects refer to this as the big "S") can impose an insurmountable number of complexities on an application's domain model, and simultaneously reduces performance to a significant degree. So what's an IT manager to do, wait and hope for the best?

Enter clustering at the JVM-level. Access to the network when reading and writing objects to heap can be transparent and efficient. Think "network attached memory" with performance optimizations that, at runtime, pushes only heap-level deltas and only to JVMs that are actively accessing particular parts of objects. With heap-level replication, the Open Source frameworks don't have to change when an enterprise IT management team decides an application is too important to run without HA, for example.

Let's examine a use case. Spring Web Flow or Rife Continuations both aim to solve a similar problem for the developer of Web applications. This is a well-understood problem around making support for the "forward" and "back" buttons in Web browsers work in a consistent and logical fashion across Web sites without custom coding by each application developer in each enterprise. For example, in visiting three Web sites a consumer adds an item to his shopping basket at each site. On one site, clicking the "back" button implicitly removes the item from the basket while on the other two it doesn't. This is due to the fact that without container support for the "back" button, a Web app developer is left to his own devices

Ari Zilka is the founder and CTO of Terracotta.

ari@terracottatech.com

> "Clustering at the JVM-level requires no code changes, so no code in frameworks like Spring, Rife, Tomcat, and Geronimo has to change and HA can be injected at runtime after an application is designed, written, QA'd, debugged, and launched into production"

> Terracotta provides a (pure Java software-based) server that acts as a network attached memory hub and efficiently brokers all communication among the application JVMs"

when defining the "back" behavior for the site. Continuations provide an easy way to restore an application's state to the mode it was in when the Web page was first rendered. This lets the developer decide what changes to state should or shouldn't be rolled back without lots of custom code.

This is a very powerful development paradigm. But it trades off the value originally sought after with load balancers and session clustering. That value was one of total fault tolerance; with sessions clustered and a sticky load balancer, no one app server was critical to production operation, and yet the applications didn't bottleneck prematurely on network replication of the session state.

The application server's internal clustering can't always guarantee that the information that the Spring or Rife framework needs to restore state is available on the server that the load balancer has sent the HTTP request to. If Serialization was used to copy the state of a Continuation around the application server cluster, all objects that can be accessed and edited during a response to that request would have to be serializable, and the developer would have to resolve possible conflicts when objects start getting duplicated around the cluster. So Spring and Rife have chosen to leave this problem to a JVM-level clustering solution and have been cooperating with Terracotta. It's a unique marriage, but it works.

Terracotta provides a production-proven JVM clustering engine for Java applications in production. Since clustering at the JVM-level requires no code changes, the technology allows Spring, Rife, Tomcat, Geronimo, and other frameworks to be clustered with no changes either to the framework or the code that enterprises write when using those frameworks. Clustering and, thus, HA, can be injected at runtime after an application is designed, written, QA'd, debugged, and launched into production, as long as no one tries to cluster that application by hand.

Terracotta provides a (pure Java software-based) server that acts as a network attached memory hub and efficiently brokers all communication among the application JVMs. It can be downloaded and comes pre-configured for many current frameworks (with more to come on the horizon). When Terracotta servers are used underneath Open Source frameworks to inject HA at runtime, the Terracotta server's production license is free. However, just to avoid any confusion, please note that Terracotta is a commercial company that offers support and services in conjunction with production use of its products.

The value is in delivering HA to any and all Open Source. And soon Terracotta will introduce a developer area on their web site, where Open Source framework integration will be fully supported via online tools consistent with the community's expectations: support forums, bug tracking systems, etc. Furthermore, the site's developer area will offer open sourced source code that implements key design patterns for working optimally with network attached memory constructs.

The key benefits of combining Open Source and Terracotta:

- Provides enterprise-level HA to Open Source
- No changes to frameworks or business logic in the presence of JVM-level clustering.
- The implementation is cost-efficient with a support-only cost model.

Enterprises can now have Open Source and fault tolerance too.

Collecting Financial Market Data with Java 5

by Michael Poulin

Decouple data acquisition from data processing

In this article I'll share my experience in using the new features in Java 5 for solving an old industry problem, the problem of collecting constantly published financial data in reliable way. The business case example I'm going to discuss relates to the acquisition of some sort of market data published by a financial data source system like Reuters.

Data Source Publishing Conditions

Assume a *Data Source* allows for a consumer subscription that results in publishing market data messages via a *DataFeed Channel*. The consumer can listen to the messages, get them from the *Event Queue,* and process them one at a time.

The *Data Source* usually provides quite a high speed in message publishing. If a consumer processes messages slower than they arrive in the *Event Queue,* the *Data Source* can temporary store not-yet- consumed messages in its buffer – *Temporary Event Storage*. However, if the buffer overflows, the *Data Source* cancels the subscription and loses all sent but not consumed data messages. In such a case, the consumer has to resubscribe to get new data. The basic data flow is shown in the diagram in Figure 1.

Michael Poulin works as a consulting technical architect for leading financial firms. He is a Sun Certified Architect for Java Technology. For the past several years, Michael has specialized in distributed computing, SOA, and application security.

m3poulin@yahoo.com

Typical Consumer Conditions

In the diagram, the consumer's *Message Receiver* operates as a listener to the data messages. To process the message, the *Message Receiver* creates a *Data Processor* object that extracts data from the message. Gathering financial information, we're dealing with highly valuable and important data, i.e., our primarily goal is to save received data as soon as possible. This is why the diagram shows only the data store – the *Database* – and an optional *Messaging System* that might be used for publishing notifications about any new data received. Obviously, the format of the data sent out by the *Data Source* differs in the most of cases from the receiver's internal format.So, minimal initial data processing in the Data Processor might involve a data format transformation, data persisting and optional notification.

As you've probably noticed, the *Message Receiver* works in a single thread mode. Since data processing includes operations in the database and messaging system, a single threaded mode constitutes high processing risks for the consumer. Those risks include:

- An inability to keep up with the speed of the arriving data, which leads to
  - *Temporary Event Storage* overflow and
  - losing data and the subscription
- Any unpredicted hanging problems in the network and the used resources
- Temporary resource unavailability.

The solution is supposed to minimize these risks.

Designing the Solution

We can minimize the risks if we decouple the data acquisition procedure from the data processing procedure. Moreover, since data processing is a repeatable procedure, we might want to process several messages in parallel. Java 5 offers a great new tool for this task – a *ThreadPoolExecutor* API and its companions are available in the *java.util.concurrent* package.

While this article is not a presentation of the *ThreadPoolExecutor*, we need to mention a few features used in the solution. The *ThreadPoolExecutor* is not really a pool of objects as a Object Pool pattern is usually understood. In particular, it consists of two major functional parts – a queue of *Runnable* objects and an execution engine. The latter starts a number of *Runnable* objects – threads – and keeps this number actual. All threads created are non-daemon threads; they belong to the same thread group and have the same NORM_PRIORITY priority.

The *ThreadPoolExecutor* maintains the corePoolSize and maximumPoolSize characteristics of the pool. When a new task is submitted and the number of running threads is less than corePoolSize, a new thread is created. As the specification states, "If there are more than corePoolSize but less than maximumPoolSize threads running, a new thread will be created only if the queue is full."

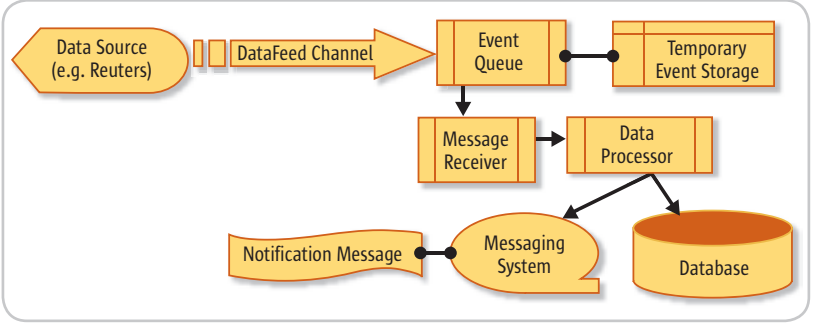


Figure 1 Basic data flow in a financial data collection

Rich Internet Applications: AJAX, Flash, Web 2.0 and Beyond...

www.AjaxWorldExpo.com

AJAXWORLD™ EAST
CONFERENCE & EXPO

NEW YORK CITY

THE ROOSEVELT HOTEL LOCATED AT MADISON & 45th

SYS-CON Events is proud to announce the AjaxWorld East Conference 2007!

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

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

BEING HELD APRIL 2-3 2007!

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

In our solution, we are interested in as much decoupling between two processes as possible. So we have to let the queue grow to avoid denying the new *Runnable* object submitted by the *Message Receiver* but we have finite memory resource; we have to process as many tasks (threads) in parallel as possible but we have to manage a number of concurrent threads in the system since they're quite delicate resources. Such conflicting requirements may be mostly satisfied if we use a queue of fixed size but big enough to have time to catch consistent queue growth and compensate it. At the same time, the maximumPoolSize has to be set to the value, which still allows its increase in reasonable boundaries (via explicit management actions at runtime). The latter may help slow down the queue growing or even stabilize it.

Plus we can engage multiple subscribers (*Message Receivers*) and split the incoming messages into a number of sub-flows. Each sub-flow of data acquisition might be organized as described above. One possible alternative solution is to use the so-called Reliable Messaging technique and temporary store received data in the messaging infrastructure until it can be permanently persisted.

Finally, the *ThreadPoolExecutor* provides an API for getting some basic pool state statistics, such as:

- current pool size
- number of completed tasks
- number of active tasks
- total amount of tasks for its lifecycle
- largest size of the pool (queue) for its lifecycle

If your application uses another Java 5 feature – JMX technology – you can easily monitor the state of the pool, i.e., the dynamics of your task processing and, in some cases, manage the pool configuration "on the fly."

The diagram in Figure 2 refines the design represented in Figure 1 by using the *ThreadPoolExecutor*. In particular, while the *Message Receiver* retrieves messages from the *EventQueue* sequentially it wraps them by *Runnable* objects – the *Data Transformers* – and passes them to the *ThreadPoolExecutor* instead of processing them immediately.

The *ThreadPoolExecutor* operates on the *Runnable Data Transformers*, i.e., it executes a corePoolSize number of threads and queues all extras. Each *Data Transformer* invokes a *Data Processor* to massage data messages as mentioned in the Typical Consumer Conditions section. Thus, the *Message Receiver* and *Data Processor* are totally decoupled and can work at their own pace.

An additional question concerns the *Data Processor*: is it shared between concurrent threads or does it run in multiple instances, one per thread. If *Data Processor* were just massaged data, I would design it in a thread-saved manner and share it between the threads. However, in our case, the *Data Processor* invokes two external resources and we have to consider a certain policy for resource connections. If a connection is shared, we develop an extra risk of serializing *Data Processor* requests for the connection, a risk of performance degradation as well as a risk of connection failure that affects all waiting *Data Processor* threads. A better, less risky design would be if we decoupled the *Data Processor* from actual connections via a Connection Pool(s). In this case, the connections can be reused but not shared and, once again, resource access becomes more transparent and suitable for our management.

The extra features shown in Figure 2 are a *Management Component* and a *Task Execution Monitor*. The *Management Component* is a standard JMX MBean that registers with a JMX server and can accept its commands to change the *ThreadPoolExecutor* configuration at runtime. The *Task Execution Monitor* thread periodically reads the pool's state (statistics) and uses the *Management Component* to broadcast them to all who might be interested, in particular, to the same JMX server. If the JMX server is equipped with an administration console and/or mechanisms that can send out different types of notification messages, your Operation Tem may be able to monitor the data collection process constantly.

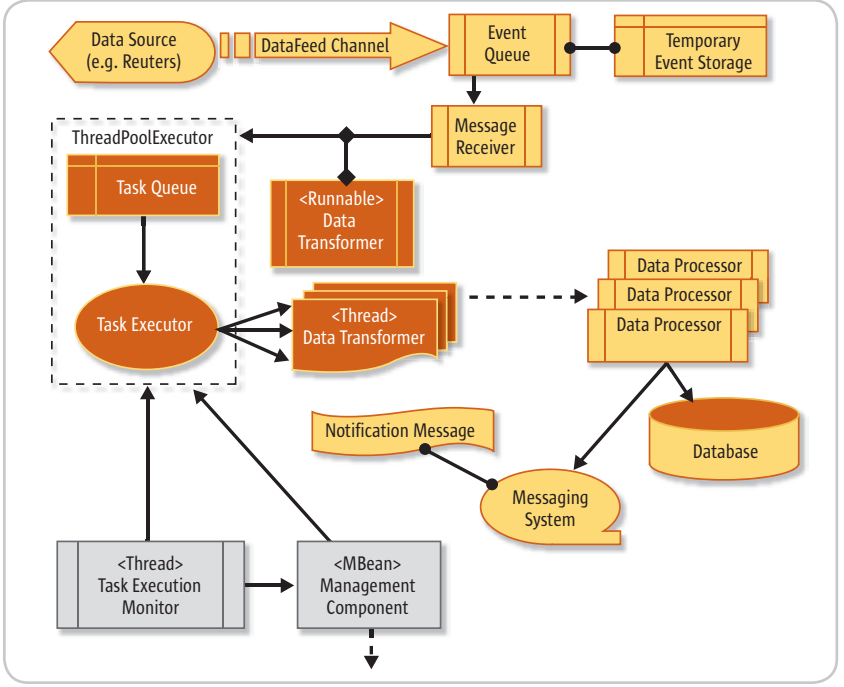


Figure 2 Decoupling data acquisition from data processing using ThreadPoolExecutor

Conclusion

We've demonstrated how Java 5's *ThreadPoolExecutor* can be used to decouple data acquisition from the data manipulation processes. Features in the *ThreadPoolExecutor* can mitigate the data processing risks, improve performance, and increase scalability in data processing. Another Java 5 feature – JMX technology – helps in monitoring and managing the *ThreadPoolExecutor* component in real-time.

References

- Class ThreadPoolExecutor, http://java.sun.com/j2se/1.5.0/docs/api/java/util/concurrent/ThreadPoolExecutor.html.
- Java Management Extensions (JMX), http://java.sun.com/products/JavaManagement/index.jsp.
- Michael Poulin. "Assured Delivery of Audit Data with SOA and Web Services." WLDJ, Volume 4 Issue 6. http://jdj.sys-con.com/read/169336.htm.

Visit the *New*

www.SYS-CON.com

Website Today!

The World's Leading *i*-Technology News and Information Source

24/7

FREE NEWSLETTERS
Stay ahead of the i-Technology curve with
E-mail updates on what's happening in your industry

SYS-CON.TV
Watch video of breaking news, interviews with industry leaders, and how-to tutorials

BLOG-N-PLAY!
Read web logs from the movers and shakers or create your own blog to be read by millions

WEBCAST
Streaming video on today's i-Technology news, events, and webinars

EDUCATION
The world's leading online i-Technology university

RESEARCH
i-Technology data "and" analysis for business decision-makers

MAGAZINES
View the current issue and past archives of your favorite i-Technology journal

INTERNATIONAL SITES
Get all the news and information happening in other countries worldwide

JUMP TO THE LEADING i-TECHNOLOGY WEBSITES:

IT Solutions Guide
Information Storage+Security Journal
JDJ
Web Services Journal
.NET Developer's Journal
LinuxWorld Magazine
Linux Business News
Eclipse Developer's Journal
MX Developer's Journal
ColdFusion Developer's Journal
XML Journal
Wireless Business & Technology
Symbian Developer's Journal
WebSphere Journal
WLDJ
PowerBuilder Developer's Journal

Joe Winchester
Desktop Java Editor

The Perils of Abstraction

Abstraction, as defined on dictionary.com, is "considering something as a general quality or characteristic, apart from concrete realities, specific objects, or actual instances." It's a powerful concept that underpins software reuse. When you implement a problem, if, instead of starting from scratch, the scenario can be thought of as being an example of an already-understood question, its solution can benefit from existing implementations.

Abstraction is a powerful concept, but it carries dangers as well. The first is those who become so enamoured with the idea of generality that they design with the goal of re-use and framework construction alone, rather than remaining focused on the concrete problem at hand. The second problem occurs when said folk have their abstract solution complete, they feel compelled to force it on every implementation that comes within range.

In a project I once worked on, a group of eager young business analysts were given the task of designing a new insurance system. The business model behind insurance is pretty simple: the insured party is quoted a policy that involves them paying you a premium in exchange for which you, the insurer, underwrite various circumstances that, should they occur, cause some kind of loss to the insured. The insurer's role is to recompense the insured for their misfortune.

The boffins designing our system decided that this was merely an instance of the more general process of "money exchanging hands for goods and services." After they parked themselves in conference rooms with walls plastered with meaningless diagrams and charts, they emerged having decided that they would design a grand and general-purpose solution for all financial transactions. This "panacea" of theirs would not only handle every possible type of insurance policy known to mankind, but it would be customizable to all other scenarios that involved money changing hands, such as banking, accounting, and electronic point of sale. The end result was a system that, while an award-winning work of art for abstraction and vagueness, failed to do the basics of insurance without having to bump and fight its way through the lower layers, delivering poor performance and a badly fitting user experience.

As the cause of such overzealous design I wonder whether programmers have an atavistic desire to find some kind of ultimate software truth. Much of twentieth-century physics was dedicated to such theorem, consolidating first magnetism and electricity before moving onto gravity. Grand unification attempts occur in other disciplines – mathematicians attempting to reduce all number theory to fundamental and irreducible truths or the biologists' desire to classify living things into taxonomical trees and genus. Do software architects feel compelled to follow this scientific path, looking for shapes in the dark or patterns in the clouds where none exist?

The second danger posed by the uber abstraction crowd is that having designed their perfect solution, they now need to nurture and promote their baby, wielding their shiny hammer at every screw, bolt, or rivet that comes within range.

"Aha, you're building a JMS server. That's just a message protocol; I already have one of those that can handle everything, so all you have to do is adapt to me and write a wrapper to my API."

The problem with this solution is that, as an implementer of the abstract framework, you have to wrestle and bridge the impedance mismatch. Your code is now concerned with how to provide a JMS interface on something that was built and optimized for another kind of message protocol. Through loss of fidelity, the end result looks and behaves like a race horse wearing rollerblades and fed with gasoline. It does the job of moving on four wheels, but clumsily and without the reliability and grace of an internal combustion engine–powered car that the original spec called for. Examples of such applications occur all the time, from those who believe that e-mail is merely a type of document for which all their singing, dancing, jumbo jet document management software can be tweaked to have an inbox and outbox, through the "I love XML" bumper sticker brigade who believe that any kind of data sent over a wire should be a W3C-compliant XML document object model when simple serialization or a basic text message would have sufficed.

For the user of the application, just as the rollerblading horse is likely to neigh from time to time, behavior and functionality from the underlying abstract layers bubble to the surface. Your messaging application throws SAX parser errors at you when things go wrong, or your e-mail product tells you that document variables aren't set correctly. The terminology of the thing the user is concerned about, the message or the e-mail, is lost as one of the layers of abstract framework code that underpins their application rears its ugly head. Joel Spolsky coins this kind of behavior Leaky Abstraction (http://www.joelonsoftware.com/articles/LeakyAbstractions.html). No matter how much wallpaper or perfume the developer used to massage and beat the abstract framework into shape for your application's implementation, at some point the abstract layers are going to rear their head as the horse needs to poop.

Alongside the opening dictionary.com definition of abstraction, which proclaims the benefits of generality, is an ironically appropriate alternative usage: "an impractical idea; something visionary and unrealistic."

Software should be built with the goal of solving a specific user scenario. In building the solution, you should make the overriding goal high-performance combined with fitness for purpose. By using as few underlying layers as possible, the number of project and physical dependencies should be kept to a minimum. When you're a hammer everything looks like a nail, yet when you're a software developer everything should look like a fresh challenge, not a problem to be short-changed by hacking some other problem's solution to fit.

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

joewinchester@sys-con.com

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

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

SYS-CON MEDIA

ONLY $69.99

ONE YEAR
12 ISSUES

Subscription Price Includes FREE JDJ Digital Edition!

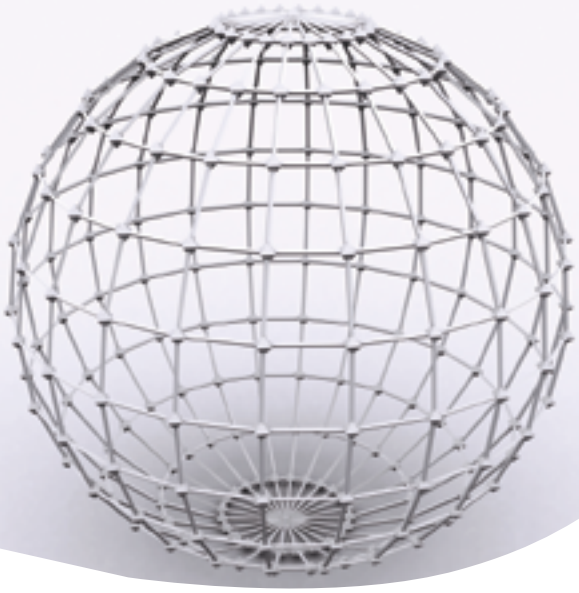
www.JDJ.SYS-CON.com

or **1-888-303-5282**

OFFER SUBJECT TO CHANGE WITHOUT NOTICE

Developing an Eclipse BIRT Report Item Extension

Architecture and framework



by Jason Weathersby, Iana Chatalbasheva, and Tom Bondur

The Eclipse platform is an Open Source, integrated system of application development tools that you implement and extend using a plug-in interface. Eclipse provides a set of core plug-ins that configures the basic services for the platform's framework. A platform developer can build and integrate new tools in this application development system.

Business Intelligence Reporting Tool (BIRT) is a set of plug-in extensions that lets a developer add reporting functionality to an application. The APIs in BIRT define extension points that let the developer add custom functionality to the BIRT framework.

This article describes how to create a BIRT extension using the Eclipse Plug-in Development Environment (PDE). The example adds a custom report item, RotatedLabel, to the BIRT Report Designer Palette that a report developer can drag-and-drop into a report design, as shown in Figure 1.

The sample code for the plug-in creates a label in the runtime report that renders text at a specified angle. Figure 2 shows the display text rotated at an angle of 45 degrees.

A developer uses the Eclipse PDE to create, develop, test, debug, and deploy a plug-in. The Eclipse PDE supports host and runtime instances of the workbench project. The host instance provides the development environment. The runtime instance lets you launch a plug-in to test it.

To implement the report item extension, the plug-in developer does the following tasks:

- Configures the plug-in project in the Eclipse PDE
- Adds the report item to the Report Designer Palette using the report item UI extension point.
- Adds the report item definition to the Report Object Model (ROM) using the report item model extension point.
- Adds rendering behavior to the report item using the report item presentation extension point.
- Deploys the report item extension to the Eclipse plug-in environment

An Eclipse plug-in implements the following components:

- ***Plug-in manifest*** – An XML document that describes the plug-in's activation framework to the Eclipse runtime environment
- ***Plug-in runtime class*** – A Java class that defines the methods for starting, managing, and stopping a plug-in instance
- ***Extension-point schema definition*** – An XML document that specifies a grammar that you must follow when defining the elements of a plug-in extension in the Eclipse PDE

In the Eclipse PDE Workbench, the developer can create the framework for a plug-in extension by using the Manifest Editor to generate the plug-in manifest and class templates based on the definitions in the extension-point schemas.

In the Eclipse PDE, create a new project for the rotated label report item extension by choosing File->New->Project and selecting the Plug-in Project wizard. In Plug-in Project, modify the settings, as shown in Table 1.

In Plug-in Content, modify the settings, as shown in Table 2.

Figure 3 shows the host instance of the Eclipse PDE with the rotated label report item extension project open in the Manifest Editor.

After defining the plug-in project, specify the list of plug-ins that must be available on the classpath of the rotated label report item extension to compile and run. On PDE Manifest Editor, choose Dependencies. In Required Plug-ins, remove the following plug-ins:

- org.eclipse.ui
- org.eclipse.core.runtime

Section	Option	Value
Plug-in Project	Project name	org.eclipse.birt.sample.reportitem.rotatedlabel
	Use default location	Selected
	Location	Not available when you select use default location.
Project Settings	Create a Java project	Selected
	Source folder	src
	Output folder	bin
Target Platform	Eclipse version	3.2
	OSGi framework	Selected

Table 1 Settings for Plug-in Project fields

The authors are members of the extended BIRT development team at Actuate Corporation and all have backgrounds in both computer science and technical writing. Collectively, they have many years experience in technical consulting, training, writing, and publishing about reporting, business intelligence tools, and database technologies.

jweathersby@actuate.com
iana@actuate.com
tbondur@actuate.com

Choose Add. Plug-in Selection appears. In Plug-in Selection, select the following plug-ins:

- org.eclipse.emf.ecore
- org.eclipse.birt.report.designer.ui
- org.eclipse.birt.report.model
- org.eclipse.draw2d
- org.eclipse.birt.report.engine
- org.eclipse.jface.text
- org.eclipse.core.runtime
- org.eclipse.birt.core
- org.eclipse.ui
- org.eclipse.birt.core.ui

The order of the list determines the sequence in which a plug-in loads at runtime. Use Up and Down to change the loading order as necessary. The rotated label report item extension doesn't need any changes to the loading order if you select the required plug-ins in the order listed in the previous step.

On PDE Manifest Editor, choose Extensions. On Extensions, declare the extension points required to implement the rotated label report item plug-in and add the extension element details. The Eclipse PDE uses the XML schema defined for each extension point to provide the list of valid attributes and values specified for the extension elements.

To add an extension point, choose Add. New Extension appears. In Available extension points, select the plug-in that contains the extension point.

The rotated label report item extension implements the following extension points:

- org.eclipse.birt.report.designer.ui.reportitemUI registers the graphical user interface (GUI) to use for the report item extension
- org.eclipse.birt.report.model.reportItemModel specifies how to represent and persist the report item extension in the Report Object Model (ROM)
- org.eclipse.birt.report.engine.reportitemPresentation specifies how to instantiate, process, and render the report item extension

To add an extension point element right-click on an extension point such as org.eclipse.birt.report.designer.ui.reportItemLabelUI. Choose New <extension point element> to add the extension element to the project. Figure 4 shows how to select the extension point element, reportItemLabelUI, specified by the extension point, org.eclipse.birt.report.designer.ui.reportitemUI.

An XML schema specifies the following properties that identify each extension point in the runtime environment:

- ***ID*** – Optional identifier of the extension instance
- ***Name*** – Optional name of the extension instance
- ***Point*** – Fully qualified identifier of the extension point

On Extensions, these settings appear in Extension Details when an extension point is selected as shown in Figure 4.

The extension point, org.eclipse.birt.report.designer.ui.reportitemUI, specifies the following extension elements:

- ***reportItemLabelUI*** – The fully qualified name of the Java class that gets the display text for the report item component in the BIRT Report Designer
- ***model*** – ROM report item extension name that maps to this UI component

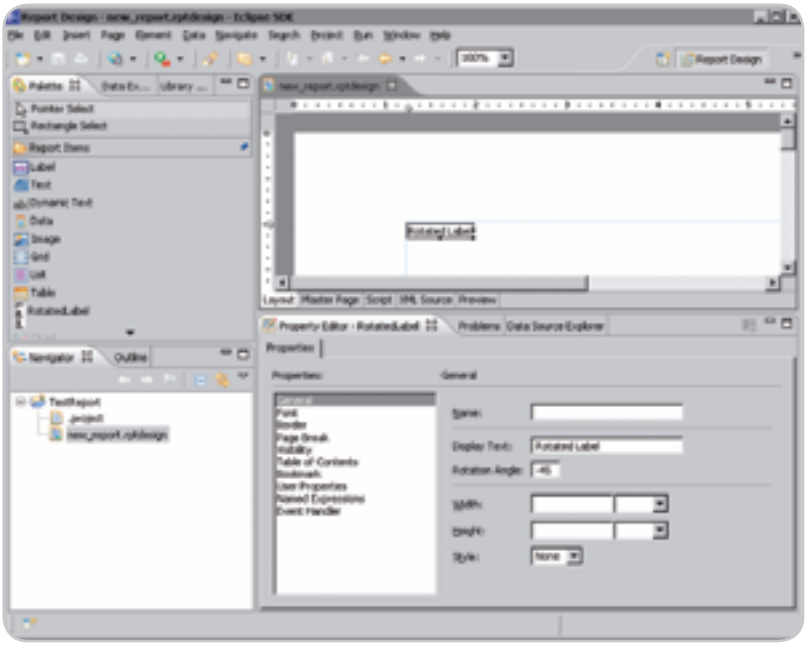


Figure 1 Rotated label report item in a report design

- ***palette*** – Icon to show and the category in which the icon appears in the Palette
- ***editor*** – Flags indicating whether the editor shows in the MasterPage and Designer UI and is resizable in the Editor
- ***outline*** – Icon to show in the Outline View
- ***propertyPage*** – Optional Property Edit Page to use for the report item extension in the Property Edit View

The extension point org.eclipse.birt.report.model.reportItemModel specifies reportItem and the following extension element properties:

- ***extensionName*** – Internal unique name of the report item extension
- ***class*** – Fully qualified name of the Java class that implements the org.eclipse.birt.report.model.api.extension.IReportItemFactory interface
- ***defaultStyle*** – Predefined style to use for the report item extension
- ***isNameRequired*** – Field indicating whether the report item instance name is required
- ***displayNameID*** – Resource key for the display name

reportItem also specifies the following property extension elements:

- rotationAngle
- displayText

rotationAngle and displayText each specify the following properties:

- ***name*** – Internal unique name of the property extension element
- ***type*** – Data type, such as integer or string
- ***displayNameID*** – Resource key for the display name
- ***canInherit*** – Flag indicating whether the property extension element can inherit properties
- ***detailType*** – Detail data type such as Boolean or string
- ***defaultValue*** – Default value of the property extension element

Figure 2 Rotated label in the report preview

Section	Option	Value
Plug-in Properties	Plug-in ID	org.eclipse.birt.sample.reportitem.rotatedlabel
	Plug-in Version	1.0.0
	Plug-in Name	RotatedLabel Plug-in
	Plug-in Provider	yourCompany.com or leave blank
	Classpath	rotatedLabel.jar
Plug-in Options	Generate an activator, a Java class that controls the plug-in's lifecycle	Selected
	Activator	org.eclipse.birt.sample.reportitem.rotatedlabel.RotatedLabelPlugin
	This plug-in	Deselected

Table 2 Plug-in content settings

- ***isEncryptable*** – Flag indicating whether the property is encrypted
- ***defaultDisplayName*** – Display name to use if no localized I18N display name exists

The extension point org.eclipse.birt.report.engine.reportitemPresentation specifies the following report Item extension elements:

- ***name*** – Unique name of the report item extension.
- ***class*** – Fully qualified name of the Java class that implements the org.eclipse.birt.report.engine.extension.IReportItemPresentation interface.
- ***supportedFormats*** – Supported rendering formats for this extended item. The value for this attribute is a comma-separated string, such as "HTML, PDF." The string is case-insensitive.

Listing 1 shows the automatically generated manifest file plugin.xml for the rotated label report item extension that describes the plug-in's activation framework to the run-time environment.

After defining the plug-in framework in the Eclipse PDE, the developer makes the code-based extensions required to complete the plug-in development process.

The rotated label report item extension implements the following interfaces and classes:

- org.eclipse.birt.report.designer.ui.extensions specifies the following interfaces:
  - ***IPropertyTabUI*** – Represents a new tab in the Property Editor view, creating the UI, updating property values on request, and notifying the BIRT framework of any UI-based property change. PropertyTabUIAdapter is the adapter class that implements this interface.
  - ***IReportItemLabelProvider*** – Defines the interface for the accessor method that provides the label text. ReportItemLabelProvider is the adapter class that implements this interface.
  - ***IReportItemPropertyEditUI*** – Provides the interface for defining tabs in the Property Editor.
- org.eclipse.birt.report.designer.ui.views.attributes.providers.PropertyProcessor provides accessor methods for processing general property information.
- org.eclipse.birt.report.engine.extension
  - ***IRowSet*** – Defines the interface to a row set. Provides metadata, grouping level, and row navigation methods.
  - ***IReportItemPresentation*** – Defines the interface for presentation of a report item extension. IReportItemPresentation sets the locale, resolution, output, and image formats and processes the extended item in the report presentation environment. ReportItemPresentationBase is the adapter class that implements this interface.
- org.eclipse.birt.report.model.api
  - ***DesignElementHandle*** – Functions as the base class for all report elements. DesignElementHandle provides a high-level interface to the BIRT report model. The class provides the generic services for all elements. Derived classes provide specialized methods for each element type. DesignElementHandle implements the interface, org.eclipse.birt.report.model.elements.interfaces.IDesignElementModel.
  - ***DesignEngine*** – Provides an interface to the BIRT design engine. DesignEngine instantiates a session handle to use when creating a new design, opening an existing design, and managing design processing. The session handle contains the report design's state. DesignEngine implements the interface, IDesignEngine interface.
  - ***ExtendedItemHandle*** – Provides a handle to an extended item that appears in a section of a report. The extended report item can have properties such as size, position, style, visibility rules, or a binding to a data source. ExtendedItemHandle extends ReportItemHandle, an abstract base class that extends DesignElementHandle.
- org.eclipse.birt.report.model.elements.Style extends org.eclipse.birt.report.model.core.StyleElement, the base class for report elements with a style, and implements org.eclipse.birt.report.model.elements.interfaces.IStyleModel, the interface for storing style element constants.
- org.eclipse.birt.report.model.api.extension specifies the following interfaces:
  - ***IMessages*** – Defines the interface for getting a localized message from a message file using a resource key.
  - ***IPropertyDefinition*** – Defines the interface for the accessor methods that describe a property. PropertyDefinition is the adapter class that implements this interface.
  - ***IReportItem*** – Defines the interface for an instance of an extended report element. There is a one-to-one correspondence between the BIRT report item and this implementation. ReportItem is the adapter class that implements this interface.

introductory subscription offer!

A TRULY INDEPENDENT VOICE IN THE WORLD OF .NET

.NET Developer's Journal is the leading independent monthly publication targeted at .NET developers, particularly advanced developers. It brings .NET developers everything they need to know in order to create great software.

Published monthly, ***.NET Developer's Journal*** covers everything of interest to developers working with Microsoft .NET technologies – all from a completely independent and nonbiased perspective. Articles are carefully selected for their prime technical content – technical details aren't watered down with lots of needless opinion and commentary. Apart from the technical content, expert analysts and software industry commentators keep developers and their managers abreast of the business forces influencing .NET's rapid development.

Wholly independent of both Microsoft Corporation and the other main players now shaping the course of .NET and Web services, ***.NET Developer's Journal*** represents a **constant, neutral, expert voice** on the state of .NET today – the good, the bad, and the ugly…no exceptions.

SYS-CON MEDIA

SUBSCRIBE ONLINE!

www.sys-con.com/dotnet/

or Call

1 888 303-5282

Here's what you'll find in every issue of .netdj:

- Security Watch
- Mobile .NET
- .NET Trends
- Tech Tips
- Standards Watch
- Business Alerts
- .NET News
- Book and Software Announcements

.NET Developer's Journal is for .NET developers of all levels, especially those "in the trenches" creating .NET code on a daily basis:

- For beginners:
  Each issue contains step-by-step tutorials.
- For intermediate developers:
  There are more advanced articles.
- For advanced .NET developers:
  In-depth technical articles and columns written by acknowledged .NET experts.

Regardless of their experience level, *.NET Developer's Journal* assumes that everyone reading it shares a common desire to understand as much about .NET – and the business forces shaping it – as possible. Our aim is to help bring our reader-developers closer and closer to that goal with each and every new issue!

SAVE 16% OFF

THE ANNUAL COVER PRICE

Get 12 issues of .NETDJ for only $69.99!

OFFER SUBJECT TO CHANGE WITHOUT NOTICE

ANNUAL COVER PRICE:

~~$83.88~~

YOU PAY

$69.99

YOU SAVE

$13.89

OFF THE ANNUAL COVER PRICE

  - ***IReportItemFactory*** – Defines the interface for the factory that creates an instance of the extended element, IReportItem. IReportItem stores the model data and serializes the model state. ReportItemFactory is the adapter class that implements this interface.
- org.eclipse.birt.report.model.metadata.PropertyType functions as the base class for the metadata of a property type. A property type provides the display name, data validation and conversion methods, XML name, and other processing. PropertyType implements the interface, org.eclipse.birt.report.model.api.metadata.IPropertyType.
- org.eclipse.core.runtime.Plugin defines the basic methods for starting, managing, and stopping the plug-in instance.

This article provides the implementation details for the most important classes in the rotated label report item extension.

For example, the RotatedLabelItemFactoryImpl class instantiates a new report item when the user drags a rotated label report item from the Palette and drops the report item in the BIRT Report Designer Editor. This class extends the adapter class, org.eclipse.birt.report.model.api.extension.ReportItemFactory.

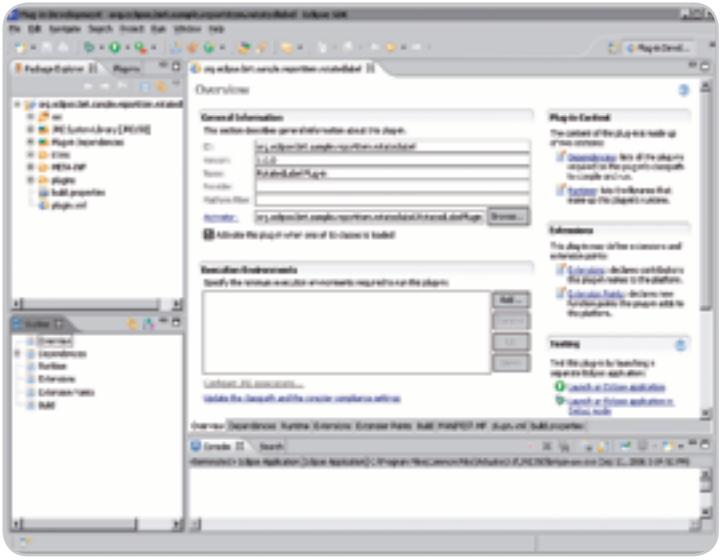


Figure 3 The host instance of the PDE Workbench

Figure 4 Adding an extension point element

In the implementation class, the newReportItem( ) method receives a reference to DesignElementHandle, which provides the interface to the BIRT report model. The newReportItem( ) method instantiates the report item, as shown in Listing 2.

In the RotatedLabelUI class, the getLabel( ) method provides the text representation for the label to BIRT Report Designer. RotatedLabelUI extends the adapter class, org.eclipse.birt.report.designer.ui.extensions.ReportItemLabelProvider. Listing 3 shows the code for the getLabel( ) method.

The RotatedLabelPresentationImpl class specifies how to process and render the report item at presentation time. This class extends the org.eclipse.birt.report.engine.extension.ReportItemPresentationBase class.

The method, onRowSets( ), renders the rotated label report item as an image, rotated by the angle specified in the report design, as shown in Listing 4.

In the RotatedLabelReportItemImpl class, the method, getPropertyDefinitions( ), instantiates RotatedLabelPropertyDefinitionImpl objects for the displayText and rotationAngle properties. RotatedLabelReportItemImpl extends the adapter class, org.eclipse.birt.report.model.api.extension.ReportItem. Listing 5 shows the code for the getPropertyDefinitions( ) method.

The RotatedLabelPropertyEditUIImpl class builds the UI using the RotatedLabelGeneralTabUIImpl class to set up the controls for the UI. RotatedLabelPropertyEditUIImpl implements the org.eclipse.birt.report.designer.ui.extensions.IReportItemPropertyEditUI interface.

In the RotatedLabelPropertyEditUIImpl class, the getCategoryTabs ( ) method instantiates the RotatedLabelGeneralTabUIImpl class, as shown in Listing 6.

The RotatedLabelGeneralTabUIImpl class contains an internal class GeneralCategoryWrapper that creates the UI contents, as shown in Listing 7.

The GraphicsUtil class creates the image containing the specified text and rotates the text image to the specified angle, using the following methods:

- createRotatedText( ) performs the following operations:
  - Gets the display text and rotation angle properties
  - Sets the display text font and determines the font metrics
  - Creates an image the same size as the display text String
  - Draws the display text as an image
  - Calls the rotateImage( ) method to rotate the image at the specified angle
  - Disposes of the operating system resources used to render the image
  - Returns the image object
- rotateImage( ) rotates the image and determines the width, height, and point of origin for the image

Listing 8 shows the code for createRotatedText( ) method.

On the PDE Manifest Editor, in Overview, the Testing section contains links to launch a plug-in as a separate Eclipse application in either Run or Debug mode. In Testing, choose Launch an Eclipse application. Eclipse launches the runtime workbench.

REALWORLD FLEX SEMINAR

www.flexseminar.com

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

August 14, 2006
The Roosevelt Hotel
New York City

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

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

"Go Beyond AJAX with Flex."

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

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

Who Should Attend?

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

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

Conference Price:
(Register Onsite) $495*

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

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

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

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

In Report Design, choose File->New->Project, and choose Report Project. Create a new report in the project by choosing File->New->Report.

In File name, type a file name if you want to change the default file name. Choose Next. New Report displays the report templates. In Report templates, choose Blank Report, and choose Finish.

The layout editor displays the report design, new_report.rptdesign. The Palette contains the RotatedText report item.

From the Palette, drag RotatedLabel to Layout, as shown in Figure 1. In new_report.rptdesign, choose Preview. The preview appears, displaying the rotated label report item, as shown in Figure 2.

This article is an excerpt from the book, *Integrating and Extending BIRT* by Jason Weathersby, Don French, Tom Bondur, Jane Tatchell, and Iana Chatalbasheva, soon to be published by Addison-Wesley. The book is the second volume in a two-book series about business intelligence and reporting technology. The book introduces programmers to BIRT architecture and the reporting framework. It shows programmers how to build and deploy customized reports using scripting and BIRT APIs. It also describes how to use key extension points to create a customized report item, a rendering extension for generating output other than HTML or PDF, and an Open Data Access (ODA) driver for a new data source. *Integrating and Extending BIRT*. Copyright 2007 Actuate. ISBN 0321443853. For more information, please visit www.awprofessional.com.

Listing 1: Plug-in manifest file

```
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.2"?>
<plugin>
   <extension
         id="rotatedLabel"
         name="Rotated Label Extension"
         point="org.eclipse.birt.report.designer.ui.reportitemUI">
      <reportItemLabelUI class="org.eclipse.birt.sample.
reportitem.rotatedlabel.RotatedLabelUI"/>
      <model extensionName="RotatedLabel"/>
      <palette icon="icons/rotatedlabel.jpg"/>
      <editor
            canResize="true"
            showInDesigner="true"
            showInMasterPage="true"/>
      <outline icon="icons/rotatedlabel.jpg"/>
      <propertyPage class="org.eclipse.birt.sample.reportitem.
rotatedlabel.RotatedLabelPropertyEditUIImpl"/>
   </extension>
   <extension
         id="rotatedLabel"
         name="Rotated Label Extension"
         point="org.eclipse.birt.report.model.reportItemModel">
      <reportItem
            class="org.eclipse.birt.sample.reportitem.rotated
label.RotatedLabelItemFactoryImpl"
            extensionName="RotatedLabel">
         <property
               defaultDisplayName="Rotation Angle"
               defaultValue="-45"
               name="rotationAngle"
               type="integer"/>
         <property
               defaultDisplayName="Display Text"
               defaultValue="Rotated Label"
               name="displayText"
               type="string"/>
      </reportItem>
   </extension>
   <extension
         id="rotatedLabel"
         name="Rotated Label Extension"
         point="org.eclipse.birt.report.engine.reportitem
Presentation">
      <reportItem
            class="org.eclipse.birt.sample.reportitem.rotated
label.RotatedLabelPresentationImpl"
            name="RotatedLabel"/>
   </extension>
</plugin>
```

Listing 2: The newReportItem( ) method

```
public class RotatedLabelItemFactoryImpl extends ReportItemFactory
implements IMessages
{
public IReportItem newReportItem( DesignElementHandle deh ) {
  return new RotatedLabelReportItemImpl( deh );
```

Listing 3: The getLabel( ) method

```
public class RotatedLabelUI extends ReportItemLabelProvider
{
 public String getLabel( ExtendedItemHandle handle )
 {
  if ( handle.getProperty( "displayText" ) != null ) {
   return ( String ) handle.getProperty( "displayText" );
  } else {
   return "Rotated Label";
  }
 }
```

Listing 4: The onRowSets( ) method

```
public Object onRowSets( IRowSet[] rowSets ) throws BirtException
 {

  if ( modelHandle == null )
  {
   return null;
  }


   // Generate the rotated text image
   graphicsUtil = new GraphicsUtil( );
```

–continued on page 52

Welcome to the Future of Video on the Web!

REGISTER NOW!
www.iTVcon.com

CALL FOR PAPERS NOW OPEN!

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

iTVCON.COM
INTERNET TV CONFERENCE & EXPO 2006

Coming in 2006 to New York City!

"Internet TV is wide open, it is global, and in true 'Web 2.0' spirit it is a direct-to-consumer opportunity!"

For More Information, Call 201-802-3023 or Email itvcon@sys-con.com

List of Topics:

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

Welcome to the Future!

Did you already purchase your ".tv" domain name?
You can't afford not to add Internet TV to your Website in 2006!

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

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

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

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

Jeremy Geelan
Conference Chair, iTVCon.com
jeremy@sys-con.com

PRODUCED BY SYS-CON EVENTS

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

```
   org.eclipse.swt.graphics.Image rotatedImage =
         graphicsUtil.createRotatedText( modelHandle );

   // Save the image to a byte array stream, via an ImageLoader
   ImageLoader imageLoader = new ImageLoader( );
   imageLoader.data = new ImageData[ ]
{ rotatedImage.getImageData( ) };
   ByteArrayOutputStream baos = new ByteArrayOutputStream( );
   imageLoader.save( baos, SWT.IMAGE_JPEG );

   return baos.toByteArray( );

 }
```

Listing 5: The getPropertyDefinitions( ) method

```
public IPropertyDefinition[ ] getPropertyDefinitions( )
{
 if ( rt == null )
 {
  return null;
 }
 return new IPropertyDefinition[ ]{
   new RotatedLabelPropertyDefinitionImpl ( null,
     "displayText", "property.label.displayText",
     false,
     PropertyType.STRING_TYPE,
     null,null,null,true),
   new RotatedLabelPropertyDefinitionImpl ( null,
     "rotationAngle",
     "property.label.rotationAngle",
     false,
     PropertyType.INTEGER_TYPE,
     null,null,null,true),
 }
}
```

Listing 6: The getCateoryTabs( ) method

```
public class RotatedLabelPropertyEditUIImpl implements IReportItem
PropertyEditUI {
 public IPropertyTabUI[ ] getCategoryTabs( ) {
  return new IPropertyTabUI[ ]{
    new RotatedLabelGeneralTabUIImpl( ),
  };
 }
}
```

Listing 7: The GeneralCategoryWrapper class

```
static class GeneralCategoryWrapper
extends AttributesUtil.PageWrapper {
 static String CATEGORY_NAME = "General";

 public void buildContent( Composite parent,
  Map propertyMap ) {
  parent.setLayout( createGridLayout( 2 ) );
  buildGridControl( parent,
    propertyMap,
    ReportDesignConstants.EXTENDED_ITEM,
    ReportItemHandle.NAME_PROP,
    1,
    false,
    new TextPropertyDescriptor
     ( new PropertyProcessor
     ( ReportDesignConstants.EXTENDED_ITEM,
        ReportItemHandle.NAME_PROP ) ),
    true,
    150);
    ...
```

Listing 8: The createRotatedText( ) method

```
public Image createRotatedText( ExtendedItemHandle
modelHandle )
{
 Image stringImage;
 Image image;
 GC gc;
  String text = "";
 if ( modelHandle.getProperty( "displayText" ) != null ) {
  text = ( String ) modelHandle.getProperty
   ( "displayText" );
 }
  Integer angle = null;
 if ( modelHandle.getProperty( "rotationAngle" ) != null ) {
  angle = ( Integer ) modelHandle.getProperty
   ( "rotationAngle" );
 }
  String fontFamily = "Arial";
 if ( modelHandle.getProperty(Style.FONT_FAMILY_PROP ) !=
  null ) {
  fontFamily = ( String ) modelHandle.getProperty
   ( Style.FONT_FAMILY_PROP );
 }
  if ( display == null ) SWT.error
   ( SWT.ERROR_THREAD_INVALID_ACCESS );

 FontData fontData = new FontData( fontFamily, 14, 0 );
 Font font = new Font( display, fontData );
 try
 {
  gc = new GC( display );
  gc.setFont( font );
  gc.getFontMetrics( );
  Point pt = gc.textExtent( text );
  gc.dispose( );
  stringImage = new Image( display, pt.x, pt.y );
  gc = new GC( stringImage );
  gc.setFont( font );
  gc.drawText( text, 0, 0 );
  image = rotateImage( stringImage, angle.doubleValue( ) );

  gc.dispose( );
  stringImage.dispose( );
  return image;
 }
 catch( Exception e )
 {
  e.printStackTrace( );
 }
 return null;
}
```

JAVA DEVELOPER'S JOURNAL Advertiser Index

Advertiser	URL	Phone	Page
.NET Developer's Journal	www.sys-con.com/dotnet	888-303-5282	55
Adobe	www.adobe.com/go/try_jdjchoice		35
AjaxWorld East Conference 2007	www.ajaxworldexpo.com	201-802-3022	47
Altova	www.altova.com	978-816-1600	4
Backbase	www.backbase.com/jsf	866-800-8996	33
Business Objects	www.businessobjects.com/devxi/misunderstood		13
IBM	ibm.com/takebackcontrol/flexible		7
Infragistics	www.infragistics.com/jsf	800-231-8588	15
Instantiations	www.instantiations.com/rcpdeveloper	800-808-3737	17
InterSystems	www.intersystems.com/cache21p	617-621-0600	11
iTVcon.com Conference & Expo	www.itvcon.com	201-802-3023	59
IT Solutions Guide	www.itsolutions.sys-con.com	888-303-5282	61
Java Developer's Journal	www.jdj.sys-con.com	888-303-5282	51
Jinfonet	www.jinfonet.com/javareporting	240-477-1000	21
Laszlo	www.openlaszlo.org		41
Northwoods Software Corp.	www.nwoods.com	800-434-9820	45
OPNET Technologies, Inc.	www.opnet.com/pinpoint	240-497-3000	19
Parasoft Corporation	www.parasoft.com/jdjmagazine	888-305-0041	Cover IV
Quest Software	www.quest.com/hero	949-754-8000	Cover II
RealWorld Flex Seminar	www.flexseminar.com		57
Software FX	www.softwarefx.com	800-392-4278	Cover III
SYS-CON Website	www.sys-con.com	888-303-5282	49
TIBCO Software Inc.	http://developer.tibco.com/	800-420-8450	27

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.

JCP: Shaping the Next Chapter

Onno Kluyt

The JCP evolves in much the same way as software: we gain experience with the current implementation, gather ideas from many sources, give an initial ordering to the many ideas, write a draft, get initial feedback, write another draft, get more feedback and so on, towards a reasonable consensus of what the next version of the product or process shall become.

No successful, sustainable efforts operate in a vacuum, and the Java Community Process is no different. Technological advancement, shifting business models, and changing markets all contribute to ever-changing environments in which our community lives. To adapt to existing changes, and/or to build responsive capabilities for the future, JCP evolves. We are at such an evolutionary turn, and along with the two Executive Committees, I am working to shape the next chapter of the community.

By the time you read this column, the process change specification will have already been submitted to JCP.org, as JSR 306. Change in the process happens by using the JCP mechanism itself, a JSR. The Executive Committees form the expert group for such a JSR with Sun as the spec lead. The process has been evolved several times, until now, in a similar manner; JSR 913 led to JCP 2.1, JSR 99 created JCP 2.5 and the current Java Specification Participation Agreement(JSPA), and JSR 215 gave us the current version JCP 2.6. You can find these on the JCP.org site - a road map of the evolution of the JCP to date.

For this iteration of the process rules, I anticipate that both the process document (http://jcp.org/en/procedures/jcp2) and the JSPA (the membership agreement and IP policy) will change. As with any JSR, the eventual specifications are subject to expert group deliberations and EC approval, but here are some of the things we will be working on.

Let's start with this question: should it be possible to implement certain specifications outside the Java platform? Yes, one can envision scenarios in which it makes sense to do this. In enterprise environments where technically different architectures need to inter-operate through Web services, service

oriented architectures, or other protocols, it can be valuable to enable this standardization work to take place directly in the JCP, or for example, to enable the implementation, in environments besides Java, of some of the specs related to XML.

Many JSRs perform Java standardization work that is based on or related to external specification work. For example, some of the Java ME JSRs relate to OSGi; there are the OMG CORBA specs in Java SE and Java EE; and the JAX* JSRs depend on work in W3C, OASIS and elsewhere. Often the spec leads and expert groups like to exchange working drafts or ideas with the working groups at these organizations. This new process JSR will explore how best to facilitate these interactions, which are called liaisons in the standards world. Initial investigation indicates that working on the JSPA confidentiality statement may address such needs, which leads me to the next group of topics that the expert group will be focusing on: transparency, duration of JSRs, and individual members.

Open source software influenced the evolution of the JCP in the past. When the JCP first started, open source software did exist, but was not generally accepted as a development method or a business model until a few years later. The Apache Software Foundation, through its expertise (and persistence!), along with many others, helped Sun understand how Java specification standardization and open source can co-exist and co-operate. Over the years, many developers and IT departments had become accustomed to using open source methods to develop and market software. The JCP adjusted to ensure that open source efforts could build compatible implementations of its specs. Since then, a new trend has been emerging: it is not only commonly accepted to use open source methodologies to develop software, but also to apply this life style to the development of specifications. This encourages us to evaluate whether community members and the general public alike have sufficient insight into the work and progress of an expert group and whether the expert group has the right tools to inform the community of its progress and design decisions.

In October 2002, JCP 2.5 was launched, enabling the compatibility of open source and Java. Since then I have seen a significant and continuing increase of individual participation in the JCP (although I just can't shake the feeling that making the membership free had some effect too). Individual members (developers participating under personal titles) are highly valued in the community. Individuals serve on the EC, various JSRs are led by individuals (among them concurrency and Groovy), and many participate on key JSRs, such as the Java SE and Java EE umbrella JSRs. The expert group will explore how best to ensure a continued, mutually effective, and efficient engagement.

As you can see from JSR 306's description on the jcp.org Web site, I proposed a rather aggressive schedule. My fellow spec leads know how hard it is to keep to such a schedule. We may not finish in May 2007, as the schedule proposes, but my goal for the year is to be as far along in the process as possible, as I know that many community members are eagerly awaiting some of the changes we are proposing here.

These are just a few of the topics that this process JSR will explore. If you read the JSR proposal on JCP.org, you'll discover more. The Executive Committee members and I are very interested to learn your opinions on the JCP and its evolution. Stay tuned for further updates on JSR progress and more news from the JCP.

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

onno@jcp.org

Complex and evolving systems are hard to test...

Parasoft helps you code smarter and test faster.

Start improving quality and accelerating delivery with these products:

Awarded **"Best SOA Testing Tool"** by Sys-Con Media Readers	**InfoWorld's 2006 Technology of the Year pick** for automated Java unit testing.	Automated unit testing and code analysis for C/C++ quality.	Memory errors? Corruptions? Leaks? Buffer overflows? Turn to…	Easier Microsoft .NET testing by auto-generating test cases, harnesses & stubs	Automate Web testing and analysis.
SOAtest™	*Jtest*™	*C++test*™	*Insure++*™	*.TEST*™	*WebKing*™

PARASOFT®

We make software work.™

Go to www.parasoft.com/JDJmagazine • Or call (888) 305-0041, x3501