JAX'09

OSGi Expert Day:
OSGi Persistence

Doug Clarke
Director of Product Management, Oracle Corp.
EclipseLink Project co-Lead
History of EclipseLink
EclipseLink Project

Java SE  |  Java EE  |  OSGi  |  Spring  |  RCP

JPA  |  MOXy  |  EIS  |  SDO  |  DBWS

Eclipse Persistence Services Project (EclipseLink)

Databases  |  XML Data  |  Legacy Systems

Copyright ©2009 Oracle Corporation
Made available under the Eclipse Public License (EPL) v 1.0 and Eclipse Distribution License (EDL) v1.0
EclipseLink and OSGi – The Plan

• Work with OSGi expert group to define OSGi persistence services blueprint
• Deliver EclipseLink as OSGi bundle(s)
• Show through examples how to leverage within an OSGi solution
• Address technical challenges as a community
OSGi Enabled Challenges

• EclipseLink Bundles
  – Bundle per component
  – Need to allow users flexibility per persistence service

• Metadata access
  – XML and Annotations in calling bundle(s)

• Resource lookup
  – JDBC, Parsers
  – User provided policy implementations
EclipseLink JPA & OSGi

• JPA 1.0
  – Need to access metadata from calling bundle
  – Need to access JDBC dynamically
  – Extensions needed to JPA for Provider flexibility
    • Service based provider solution
    • Direct usage supported as well
  – Weaving requires Equinox hooks

• JPA 2.0
  – Spec formalizes ‘resolver’ extensions
JPA 1.0 Provider Lookup

- `javax.persistence.Persistence`
  - Application Bootstrap API

```java
public static EntityManagerFactory createEntityManagerFactory(
    String persistenceUnitName, Map properties)
{
    ...
    findAllProviders();
    ...

private static void findAllProviders() throws IOException
{
    ClassLoader loader =
    Thread.currentThread().getContextClassLoader();
    Enumeration<URL> resources =
        loader.getResources("META-INF/services/");
    PersistenceProvider.class.getName());
```
JPA Bootstrapping in OSGi

• **Vendor specific bootstrapping**

```java
import org.eclipse.persistence.jpa;
import javax.persistence.*;

... 

EntityManagerFactory emf = 
    PersistenceProvider.createEntityManagerFactory("pu-name");
``` 

• **Standard JPA Application Bootstrapping**

```java
import javax.persistence.*;

... 

EntityManagerFactory emf = 
    Persistence.createEntityManagerFactory("pu-name");
```

– Requires vendor registry using services ...
```java
public static EntityManagerFactory createEntityManagerFactory(
    String persistenceUnitName, Map properties)
{
    EntityManagerFactory emf = null;
    PersistenceProviderResolver resolver =
        PersistenceProviderResolverHolder.
            getPersistenceProviderResolver();

    List<PersistenceProvider> providers =
        resolver.getPersistenceProviders();
```
EclipseLink Weaving Support

- Uses (ASM) to introduce additional functionality into the ‘POJO’ domain classes
- Used for
  - M:1 and 1:1 lazy fetching
  - Fetch Groups
  - Change Tracking
  - State Caching
- Integrated with EJB3 and Spring 2.0
- Available for Java SE platform using JDK/JRE –javaagent:
- Use is Optional (used by default when possible)
- Static weaving also supported
  - Weaving of .class files before deployment

Copyright ©2009 Oracle Corporation
Made available under the Eclipse Public License (EPL) v 1.0 and Eclipse Distribution License (EDL) v1.0
DEMO: Simple RCP using EclipseLink

Diagram:
- Comics View (RCP) -> annotated model
- annotated model -> JDBC
- Eclipse RCP
- JPA
- JPA equinox
- core
- EclipseLink

Table:
<table>
<thead>
<tr>
<th>Publisher</th>
</tr>
</thead>
<tbody>
<tr>
<td>id name</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>*</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>id name</td>
</tr>
<tr>
<td>format</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>*</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Issue</th>
</tr>
</thead>
<tbody>
<tr>
<td>id comments condition copies pricePaid</td>
</tr>
</tbody>
</table>
Future Challenges & Goals

• Investigate further bundle splitting
  – As required by usage scenarios

• Address weaving enhancements
  – Optimize Equinox hook implementation
  – Address more usage configurations
    • Classes separate from XML config files

• Usability: Documentation and Examples

• Automated Testing

• Standardize
  – Provider registration
  – Weaving solution across OSGi implementation