Building Java Persistence API Applications with Dali 1.0

Shaun Smith
shaun.smith@oracle.com

ORACLE
A little about Me

- Eclipse Dali JPA Tools Project Co-Lead
- Eclipse Persistence Services Project (EclipseLink) Ecosystem Development Lead
- Eclipse Teneo Project Committer
- Product Manager for Oracle TopLink
  - Involved with object-relational and object-XML mapping technology for over 10 years.
About Java Persistence API (JPA)

- Separate document bundled as part of EJB 3.0 specification
- Suitable for use in different modes
  - Standalone in Java SE environment
  - Hosted within a Java EE Container
- Standardization of current persistence practices
- Merging of expertise from persistence vendors and communities including: TopLink, Hibernate, JDO, EJB vendors and individuals
JPA—in a Nutshell

- A Java standard that defines:
  - how Java objects are stored in relational databases (specified using a standard set of mappings)
  - a programmer API for reading, writing, and querying persistent Java objects ("Entities")
  - a full featured query language
  - a container contract that supports plugging any JPA runtime in to any compliant container.
JPA—POJO Entities

- Concrete classes
- No required interfaces
  - No required business interfaces
  - No required callback interfaces
- `new()` for instance creation
- Direct access or getter/setter methods
  - Can contain logic (e.g. for validation, etc.)
- “Managed” by an EntityManager
- Can leave the Container (become “detached”)
Object-Relational Mappings

- Core JPA Mappings
  - Id
  - Basic
  - Relationships
    - OneToOne
    - OneToMany/ManyToOne
    - ManyToMany
  - And more…

- Can be specified using Annotations or XML
@Entity public class Customer {

@Id
private String name;
@OneToOne
private Account account;

public String getName() { return name; }
public void setName(String name) {
    this.name = name;
}
public Account getAccount() { return account; }
public void setAccount(Account account) {
    this.account = account;
}
}
JPA Entity—Mappings on Properties

@Entity
gpublic class Customer {

    private String name;
    private Account account;

    @Id
    public String getName() { return name; }
    public void setName(String name) {
        this.name = name;
    }
    @OneToOne
    public Account getAccount() { return account; }
    public void setAccount(Account account) {
        this.account = account;
    }
}
JPA Entity—Mappings in XML

```xml
<entity-mappings
    xmlns="http://java.sun.com/xml/ns/persistence/orm"

...  

<entity class="Customer">

    <attributes>
        <id name="name"/>
        <one-to-one name="account"/>
    </attributes>

</entity>

...

</entity-mappings>
```
Dali—Design Time JPA Development Tools

- Support for the definition, editing, and deployment of Object-Relational (O/R) mappings for JPA Entities
- Simplify mapping definition and editing through:
  - intelligent mapping assistance
  - dynamic problem identification
  - generation and automated mapping wizards
- Extensible frameworks and tools so vendors and open source projects can provide specific support for their JPA runtimes
- A subproject of the Web Tools Platform (WTP)
Dali Goals

- **Simplicity**
  - Intelligent mapping assistance and automated generation

- **Intuitiveness**
  - Use existing and consistent modeling and tooling practices in Eclipse
  - Light-weight views offer assistance but don’t get in the way of power users

- **Compliance**
  - Support any and all JPA compliant runtime implementations

- **Extensibility**
  - Provide the ability for vendors and open source projects to seamlessly add their own value-add features
Why ‘Dali’?

JPA supports “The Persistence of Memory”—which is the title of a well known Salvador Dali painting.

© 2005 Salvador Dalí, Gala-Salvador Dalí Foundation/Artists Rights Society (ARS), New York
The Dali Value Proposition

You don’t need Dali to build JPA applications—but you’re going to be way more productive if you do!
Why do you need JPA Tools?

- JPA runtime combines:
  - Java Classes
  - Mapping Metadata
  - Database schema
Why do you need JPA Tools?

- How can you tell if they all match?
  - Deploy and run tests?
    - × slow
    - × find one problem at a time (fix, run, fix, …)
    - ✓ definitive
  - Design time validation?
    - ✓ quick
    - ✓ finds all issues
    - ✓ validates against spec
    - × runtime may not match spec

![Diagram of JPA Application Design and Runtime](attachment:image.png)
Dali Contributions to Eclipse

- JPA Mapping Validation
- JPA Structure and Details Views
- Java Source Editor enhancements
Mapping Validation

- Annotations and/or XML used to define JPA Entities.

- JDT validates syntax, but doesn’t understand what the annotations mean.
Mapping Validation

- Java Source Editor enhancements
- Mapping Problem Markers

```
@Entity
public class Address {
    @Id
    private int id;
    private String street;
    private String city;
    private String province;
    private String country;
}
```

ADDRESS

<table>
<thead>
<tr>
<th>ID</th>
<th>STREET</th>
<th>...</th>
<th>STATE</th>
</tr>
</thead>
</table>

Default mapping won’t work!

The column province cannot be found on table Address

Address.java
Mapping Assistance

- JPA Details View

```java
@Entity
public class Address {
    @Id
    private int id;
    private String street;
    private String city;
    private String province;
    private String country;
}
```

The column province cannot be found on table Address in Address.java.
Mapping Assistance for Basic Mapping

```
@Entity
public class Address {
    @Id
    private int id;
    private String street;
    private String city;
    @Column(name="STATE")
    private String province;
}
```

No Mapping Errors!
JPA Structure View

- Provides a JPA specific view of Java Class or ORM XML Mapping File
- A thumbnail sketch of how an Entity is mapped
- Supports navigation between mappings
- Automatically adjusts to either property or field mapping in Java
- Represents structure in Java and XML artifacts
JPA Perspective—all your JPA Views
Synchronizing Persistence.xml

- In Java SE environment, persistence.xml must list the Entities—Dali offers synchronization
Synchronizing Persistence.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml"
    <persistence-unit name="jdi">
        <class>model.Customer</class>
        <class>model.CustomerInfo</class>
        <class>model.Invoice</class>
        <class>model.Phone</class>
    </persistence-unit>
</persistence>
```
Integrated F1 Help

- In any Dali view you can hit F1 to get context specific help.

One-to-one mapping

Use a One-to-One Mapping to define a relationship with one-to-many multiplicity.

1. In the Persistence Outline view, select the field to map. The Persistence Properties view (for attributes) displays the properties for the selected.
2. In the Map As field, select One-to-one.
3. Use this table to complete the remaining fields on the General tab in Persistence Properties view.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target Entity</td>
<td>The entity to which this attribute is mapped.</td>
<td>Default</td>
</tr>
<tr>
<td>Cascade Type</td>
<td>See &quot;Cascade Type&quot; for details:</td>
<td>You do not need to explicitly specify the target entity, since it can be inferred from the type of object being referenced.</td>
</tr>
<tr>
<td>Fetch Type</td>
<td>Defines how data is loaded from the database.</td>
<td>Eager</td>
</tr>
</tbody>
</table>
Dali 1.0 Features

- Support for creation and edition of JPA Java annotations in source
- Support for XML Mapping File (orm.xml) configuration
  - Uses same views as JPA annotation editing
- Validation of Java annotations and XML mappings (orm.xml)
  - against object and data model (including unspecified default mappings)
  - the combination as defined by the JPA spec.
- Problems markers in Java source (e.g., Entity missing @Id)
- Integration with Data Tools Platform (DTP) for database meta-data
- JPA Details view and Java source editor support mapping using database schema with table and column drop downs
- Wizards for basic Entity generation from tables
- Extensibility for vendor specific extensions
Dali 1.0 Demo
JPA Application Development Scenarios

Possible Approaches using Dali

- Meet in the Middle
  - existing object and data models

- Bottom Up
  - generate mapped object model from data model

- Top Down
  - generate data model from mapped object model
Meet In the Middle (MITM)

- The advantage of MITM is that you can focus on getting your object model and data models correct.
  - Table ≠ Class
    - 1 Table could be N classes (using embedded)
    - N tables could be 1 class (with secondary table)
  - Use Java language features like inheritance not present in relational model

- Dali’s validation makes MITM practical
  - Avoids map, deploy, debug cycle
  - Provides access to database schema to provide valid choices
Bottom Up

- Generate Entities from Tables
  - Great way to bootstrap a JPA application from an existing database
  - Uses an Entity == Table approach
  - Do it once and then modify the generated Entities
    - Dali mapping validation will help identify issues resulting from modifications to Entities
Demo
Top Down

- Generate DDL from Entities
  - Tables will reflect Entity Mappings (including defaults)
  - Extension point in Dali to allow for plugging in runtime DDL generation.
- Typical Scenario:
  - Develop Entities with database schema disconnected (no complaints about missing schema elements)
  - Generated DDL to create necessary schema—should have no errors.
  - Modify Entity mappings—rerun DDL gen to update schema
Dali Adoption

- Oracle
  - Providing majority of resources for Dali
  - Planning to build extensions to Dali for Oracle TopLink and TopLink Essentials runtimes

- RedHat/JBoss
  - Incorporating Dali into JBossIDE as part of Hibernate/JPA toolset

- SAP
  - Shipped Dali in Eclipse based SAP NetWeaver Developer Studio Java EE 5 preview

- IBM
  - Incorporating Dali into future tooling projects
Dali 2.0 Roadmap

- Public API for Adopters to build upon
- Internal model enhancements
- Persistence.xml editing
- Named Query support
- Support for
  - Callbacks
  - IdClass
  - Multiple Persistent Unit support
  - Mapping Java Classes (with no source) with XML
  - Entity level Generators
- Quick Fixes
- ORM XML element/attribute value completion
- Support adding JPA functionality to an existing Java project
Summary

- JPA—the new Java EE standard for object-relational mapping (inside and outside of a container)

- Dali—the WTP project bringing developer productivity to JPA
  - Mapping validation to avoid the map→deploy→debug→cycle cycle
  - Intelligent mapping assistance to avoid problems and speed up the process of mapping
  - Integrated with WTP to support development for Java SE and EE
Getting Started with JPA with Eclipse

- **Dali JPA Tools**—open source tools for JPA development
  [http://www.eclipse.org/dali](http://www.eclipse.org/dali)

- **EclipseLink JPA**—Eclipse Persistence Services Project
  [http://www.eclipse.org/eclipselink](http://www.eclipse.org/eclipselink)

- **EJB 3.0 & JPA Specification**