The Yellow Brick Road to Building EJB 3.0 Applications

Mike Keith
michael.keith@oracle.com

http://www.oracle.com/technology.ejb3
About Me

- Co-spec Lead of EJB 3.0 (JSR 220)
- Java EE 5 (JSR 244) expert group member
- Co-author “Pro EJB 3: Java Persistence API”
- Persistence/EJB/Java EE Architect for Oracle
- 15+ years experience in distributed, server-side and persistence implementations
- Presenter at numerous conferences and events
About You

- How many people already know a bit about EJB 3.0?
- How many people have already built applications using EJB 3.0?
- How many people never really plan on using EJB 3.0 but are mildly interested in seeing how one would use it?
Learn how to build, configure and package basic EJB 3.0 applications
Agenda

Introduction to EJB 3.0
Building Components
Building Entities
Packaging the Components
Packaging the Entities
Other Packaging Options
Summary
About EJB 3.0

- Revolutionary and evolutionary changes
- Three main divisions of technology
  - Traditional session, MDB, entity beans (a la EJB 2.1)
  - New simplified component API for session beans
  - New POJO persistence (Java Persistence API)
- Released May 2006 as part of Java EE 5
- Configuration by exception - easier development
- Adopted practice of dependency injection (also incorporated into Java EE 5)
- Added AOP-like facility for interception
Reference Implementation

- “Glassfish” project on java.net
  - RI for entire Java EE platform
- Sun and Oracle partnership
  - Sun Application Server + Oracle persistence for JPA support
- First fully-compliant server available
  - http://glassfish.dev.java.net
- All Open Source (under CDDL license)
  - Anyone can download/use source code or binary code in development or production
EJB Components

- Traditional Components
  - Primary vehicle for business domain logic
  - Session beans, message-driven beans, EJB 2.x
  - CMP/BMP entity beans
  - Declarative container services
    - Transactions, security, concurrency
    - Managed by Container
      - Deployment, distribution, execution, access
      - Bound to the managing container
  - Traditionally required substantial metadata
EJB Components

- EJB 3.0 Simplified Components
  - Session beans, message-driven beans
  - Same declarative container services
    - Transactions, security, concurrency
  - Fewer constraints
    - Bean class does not need to implement SessionBean
    - No Home interfaces required
    - Business interfaces do not need to extend EJBOBJECT
  - Metadata may be specified in annotation form
    - May be minimal due to extensive use of defaults
JPA Entities

- Entities
  - Lightweight persistent objects (POJOs)
  - Fine-grained persistent state
    - Used by components containing the domain logic
  - Defined by Java Persistence API
  - Managed by a local EntityManager
    - Accessed through EntityManager API
    - May be detached and merged back in
  - No additional container services
    - “Inherits” container services of components that operate on them
EJB 3.0 Application Architecture

- Application
- Web Tier
- EJB and persistence metadata
- Entity
- EJB Tier
- Session Bean
- EntityManagerFactory
- Java Persistence API
- EntityManager
- RDBMS
Our EJB 3.0 Application

Application

Web Tier

EJB Tier

Flight

Passenger

Airline Reservation Bean

.persistence.xml

EntityManagerFactory

Java Persistence API

EntityManager

RDBMS
Agenda

Introduction to EJB 3.0

Building Components

Building Entities

Packaging the Components

Packaging the Entities

Other Packaging Options

Summary
Building the Session Bean

What’s in a Session Bean?

Session Bean = Business Interface + Implementation class + Metadata

- One or more business interfaces (local or remote)
- Concrete POJO implementation class
- Implementation class typically implements interface(s)
- Minimal metadata in annotation or XML form
Building the Session Bean

Business Interface

```java
public interface AirlineReservation {

    public void addFlight( ... );

    public void addPassenger( ... );

    public boolean bookFlight( ... );

    public Collection getAllFlights ( ... );

    public Collection getFlightsForPassenger( ... );

    ...

}
```
Building the Session Bean

Implementation Class

```java
public class AirlineReservationBean implements AirlineReservation {

    EntityManager em;

    public void addFlight( … ) {
        Flight f = new Flight( … );
        em.persist(f);
    }

    public void addPassenger( … ) {
        Passenger p = new Passenger( … );
        em.persist(p);
    }
}
```
Building the Session Bean

Implementation Class

```java
public boolean bookFlight(int passId, int flId) {
    Flight f = em.find(Flight.class, flId);
    Passenger p = em.find(Passenger.class, passId);
    if ((f == null) || (p == null))
        return false;
    if (f.getPassengers().contains(p))
        throw new RuntimeException("Already booked");
    f.addPassenger(p);
    return true;
}
```
public Collection getAllFlights() {
    return em
        .createNamedQuery("Flight.findAll")
        .getResultList();
}

public Collection getFlightsForPassenger(
        int passId) {
    return em
        .createNamedQuery("Flight.findByPassengerId")
        .setParameter("id", passId)
        .getResultList();
}
Adding the Metadata

Annotation Metadata

```java
@Stateless
public class AirlineReservationBean
    implements AirlineReservation {

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

    public void addFlight( ... ) {
        ...
    }

    ...
}
```
Adding the Metadata

XML Metadata

```xml
<ejb-jar xmlns="http://java.sun.com/xml/ns/javaee" version="3.0">
  <session>
    <ejb-name>AirlineReservationBean</ejb-name>
    <ejb-class>AirlineReservationBean</ejb-class>
    <session-type>Stateless</session-type>
  </session>

  <!-- (without the persistence context ref) -->
</ejb-jar>
```
Adding the Metadata

Using XML for Injection

```xml
<injection-target>
  <injection-target-class>AirlineReservationBean</injection-target-class>
  <injection-target-name>em</injection-target-name>
</injection-target>
<persistence-context-ref>
  <persistence-unit-ref-name>jpa/EM</persistence-unit-ref-name>
  <persistence-unit-name>AirlineReservation</persistence-unit-name>
</persistence-context-ref>
```
Agenda

Introduction to EJB 3.0
Building Components
Building Entities
Packaging the Components
Packaging the Entities
Other Packaging Options
Summary
Building the Entities

What’s in an Entity?

Entity = Implementation class + Metadata

- Concrete POJO implementation class
- No business interface required (but may be used if desired)
- Minimal metadata in annotation or XML form
Building the Entities

Passenger Implementation Class

```java
public class Passenger {

    int id;
    String name;
    Collection<Flight> flights;
    ...

    public void addFlight(Flight flight) {
        this.flights.add(flight);
    }

    // Rest of the getter/setter methods, etc.
}
```
Building the Entities

Flight Implementation Class

public class Flight {

    int id;

    String destination;
    String source;

    Timestamp departure;
    Timestamp arrival;

    Collection<Passenger> passengers;

Building the Entities

Flight Implementation Class

```java
public Flight() {}
public Flight(int id, String source, String dest,
              Timestamp dep, Timestamp arr) {
    this.id = id;
    this.source = source;
    this.destination = dest;
    this.departure = dep;
    this.arrival = arr;
    this.passengers = new ArrayList<Passenger>();
}
public void addPassenger(Passenger p) {
    this.passengers.add(p);
    p.addFlight(this);
}
...```

Mike Keith — The Yellow Brick Road to Building EJB 3.0 Applications
Adding the Mapping Metadata

Annotation Metadata

```java
@Entity
public class Passenger {

    @Id
    int id;

    String name;
    @ManyToMany(mappedBy="passengers")
    Collection<Flight> flights;

    ...

}
```
Adding the Mapping Metadata

Annotation Metadata

```java
@Entity
public class Flight {

@Id
int id;
@Column(name="DEST")
String destination;
String source;

Timestamp departure;
Timestamp arrival;
@ManyToMany
Collection<Passenger> passengers;

...
Adding the Mapping Metadata

Query Annotation Metadata

```java
@NamedQueries(
    @NamedQuery(name="Flight.findAll",  
        query="SELECT f FROM Flight f"),
    @NamedQuery(name="Flight.findByPassengerId",  
        query="SELECT f FROM Flight f JOIN f.passengers p "        + "WHERE p.id = :id"))

public class Flight {
    ...
}
```
Adding the Mapping Metadata

XML Mapping Metadata

- Annotations on entities are not required
- Can put the mapping metadata in one or more XML mapping files
  - Decouples mapping metadata from entities
  - May distribute mapping metadata across annotations and mapping files
  - Default mapping file named orm.xml will be automatically detected if present in META-INF
Adding the Mapping Metadata

XML Mapping File

<entity-mappings
   xmlns="http://java.sun.com/xml/ns/persistence/orm"
   version="1.0">

   <entity class="Passenger">
      <attributes>
         <id name="id"/>
         <many-to-many name="flights"
            mapped-by="passengers"/>
      </attributes>
   </entity>

</entity-mappings>
Adding the Mapping Metadata

XML Mapping File

```xml
<entity class="Flight">
  <attributes>
    <id name="id"/>
    <basic name="destination">
      <column name="DEST"/>
    </basic>
    <many-to-many name="passengers"/>
  </attributes>
</entity>
</entity-mappings>
```
Adding the Mapping Metadata

Queries in XML Mapping File

```xml
<entity-mappings ... version="1.0">
  <named-query name="Flight.findAll">
    <query>SELECT f FROM Flight f</query>
  </named-query>
  <named-query name="Flight.findByPassengerId">
    <query>SELECT f FROM Flight f JOIN f.passengers p WHERE p.id = :id</query>
  </named-query>
  <entity class="Passenger"> ... </entity>
</entity-mappings>
```
Adding the Mapping Metadata

XML Mapping Metadata

- Can also set default values
  - For all of the entities in the persistence unit
  - For only the entities listed in the mapping file

```xml
<persistence-unit-metadata>
  <persistence-unit-defaults>
    <schema>ALASKAIR.DATA</schema>
    <cascade-persist/>
  </persistence-unit-defaults>
</persistence-unit-metadata>
```
Culture Break – Beavers

- National symbol of Canada and state symbol of 2 U.S. states
- Second only to humans in ability to change their environment
Agenda

Introduction to EJB 3.0
Building Components
Building Entities
Packaging the Components
Packaging the Entities
Other Packaging Options
Summary
Packaging the Components

Creating the EJB JAR

- The EJB components go in an EJB JAR file that contains
  - Business interface(s)
  - Implementation classes

```
airlineEjb.jar

AirlineReservation.class
AirlineReservationBean.class
```
Packaging the Components

Using XML

- May optionally use XML deployment descriptor
  - Create ejb-jar.xml file and add XML metadata
  - Add to META-INF directory in EJB JAR file
  - Do not need annotations in bean classes

**airlineEjb.jar**

- AirlineReservation.class
- AirlineReservationBean.class
- META-INF/ejb-jar.xml

**ejb-jar.xml**

```xml
<ejb-jar ...version="3.0">
  <session>
    ...
    </session>
</ejb-jar>
```
Agenda

Introduction to EJB 3.0
Building Components
Building Entities
Packaging the Components
Packaging the Entities
Other Packaging Options
Summary
Packaging the Entities

Persistence Unit Metadata

- Create persistence unit metadata in XML file called persistence.xml
  - Names the persistence unit and defines its configuration
  - Defines the data source where entities are stored
  - Vendor properties for additional configuration
  - Other optional metadata if needed
    - Persistence provider class (if non-default used)
    - Additional ORM mapping files
    - JAR files of additional entities
Packaging the Entities

persistence.xml File

```xml
<persistence>
    <persistence-unit name="AirlineReservation">
        <jta-data-source>jdbc/OracleDB</jta-data-source>
        <properties>
            <property name="toplink.ddl-generation" value="create-tables"/>
            <property name="toplink.logging.level" value="FINEST"/>
        </properties>
    </persistence-unit>
</persistence>
```
Packaging the Entities

Entities in the EJB JAR

- Best to package entities in EJB JAR with the components
  - Add entity classes
  - Add persistence.xml file to META-INF directory

```
airlineEjb.jar
```

```
AirlineReservation.class
AirlineReservationBean.class
Flight.class
Passenger.class
META-INF/persistence.xml
```
Packaging the Entities

Using XML Mapping Files

- Can override annotated or defaulted mappings
  - Facilitates different database platform/schema

```xml
<entity class="Flight">
  <attributes>
    <basic name="departure">
      <column name="DEP"/>
    </basic>
  </attributes>
</entity>
```
Packaging the Entities

Packaging the Mapping File

- Default orm.xml file in META-INF directory
  - Always detected (even if other mapping files added)
  - Existing annotations will also be processed

```
<entity-mappings
  ...version="1.0">
  <entity>
    <entity>
      ...
    </entity>
    ...
  </entity-mappings>
```
Packaging the Entities

Packaging Multiple Mapping Files

- Sometimes use mapping file for each class
- Additional mapping files can have any name
  - Must be on the classpath of the module
  - Must be explicitly listed in persistence.xml

```xml
<persistence-unit>
    ...
    <mapping-file>flight.xml</mapping-file>
    <mapping-file>passenger.xml</mapping-file>
    ...
</persistence-unit>
```
Packaging the Entities

Packaging Multiple Mapping Files

**airlineEjb.jar**
- AirlineReservation.class
- AirlineReservationBean.class
- Flight.class
- Passenger.class
- META-INF/persistence.xml
- flight.xml
- passenger.xml

**flight.xml**

```xml
<entity-mappings
    ...version="1.0">
  <entity class="Flight">
    ...
  </entity>
</entity-mappings>
```

**passenger.xml**

```xml
<entity-mappings
    ...version="1.0">
  <entity class="Passenger">
    ...
  </entity>
</entity-mappings>
```
Agenda

Introduction to EJB 3.0
Building Components
Building Entities
Packaging the Components
Packaging the Entities
Other Packaging Options
Summary
Other Packaging Options

Persistence Archive

- Can create a separate persistence archive
  - Put all of the persistence artifacts in the JAR
  - Put archive in library directory in the application EAR
  - Archive is shared amongst all application modules

airlineEntities.jar

- Flight.class
- Passenger.class
- META-INF/persistence.xml
- META-INF/orm.xml
Other Packaging Options

Airline Application

airline.ear

airline.war

AirlineServlet.class

Flight.class

Passenger.class

META-INF/persistence.xml

airlineEjb.jar

AirlineReservation.class

AirlineReservationBean.class

lib/airlineEntities.jar

Flight.class

Passenger.class

META-INF/persistence.xml
Other Packaging Options

Persistence Library Using Mapping File

- Can create a **persistence library**
  - Leave persistence.xml and mapping file in EJB JAR
  - Put all of the entities in a separate JAR
  - Put JAR on classpath (library directory of the EAR)
  - Library is shared amongst all application modules

---

**airlineEjb.jar**
- AirlineReservation.class
- AirlineReservationBean.class
- META-INF/persistence.xml
- META-INF/orm.xml

**airlineEntities.jar**
- Flight.class
- Passenger.class
Other Packaging Options

Persistence Library using Annotated Mappings

- When annotations are used for mapping, the entities must be referenced in persistence.xml
  - No mapping file included in EJB JAR or library JAR
  - May be referenced explicitly by class or by library JAR
  - Classes or JAR must be on module classpath

```
<table>
<thead>
<tr>
<th>airlineEjb.jar</th>
<th>airlineEntities.jar</th>
</tr>
</thead>
<tbody>
<tr>
<td>AirlineReservation.class</td>
<td>Flight.class</td>
</tr>
<tr>
<td>AirlineReservationBean.class</td>
<td>Passenger.class</td>
</tr>
<tr>
<td>META-INF/persistence.xml</td>
<td></td>
</tr>
</tbody>
</table>
```
Other Packaging Options

Referencing Entities From persistence.xml

- By Class
  ```xml
  <persistence-unit name="AirlineReservation">
    ...
    <class>Flight</class>
    <class>Passenger</class>
  </persistence-unit>
  ```

- By JAR
  ```xml
  <persistence-unit name="AirlineReservation">
    ...
    <jar-file>airlineEntities.jar</jar-file>
  </persistence-unit>
  ```
Agenda

Introduction to EJB 3.0
Building Components
Building Entities
Packaging the Components
Packaging the Entities
Other Packaging Options

Summary
Best Practices

- Use EJB session beans for the business layer and access entities from there
- Use annotations for metadata except in cases of overriding or mapping to other databases
- If only a single EJB module in application then package everything together in the EJB JAR
- If entities are accessed by multiple modules create a shared persistence archive
- When entities are shared by multiple modules, but not always persistent use persistence library
EJB 3.0 components and entities are easy to develop, build, package and deploy.

Components and persistent entities can be packaged together or separately.

Annotations make development simpler and can facilitate practical XML-less deployment.

XML may be used instead of annotations or to override annotated configuration.

Packaging is simple for most applications but flexible enough to handle advanced requirements.
IDE Support

- Eclipse “Dali” project (http://www.eclipse.org/dali)
  - JPA support
  - Oracle (project lead), BEA, JBoss, Versant
- NetBeans (http://community.java.net/netbeans)
  - EJB 3.0 support including JPA (Beta 2)
  - Sun
- JDeveloper (http://www.oracle.com/technology/jdev)
  - EJB 3.0 support including JPA (10.1.3.1)
  - Oracle
- Developed against the Glassfish RI as the target platform
Links and Resources

- **EJB 3.0 specification**

- **Papers and Tutorials**
  - [http://glassfish.dev.java.net/javaee5/ejb](http://glassfish.dev.java.net/javaee5/ejb)
  - [http://otn.oracle.com/ejb3](http://otn.oracle.com/ejb3)

- **Books**
  - Pro EJB 3: Java Persistence API (Apress)
If you only remember one thing...

mmm... EJB 3... EJB 3