EJB - Quo Vadis?
Java Forum Stuttgart 2005

Doug Clarke

Principal Product Manager

Oracle TopLink

ORACLE

Agenda

- JSR 220 / EJB 3.0 – Overview of Changes
- Compare with EJB 2.1
- Oracle Support of EJB 3.0

EJB 3.0 Expert Group Agenda

- Define “Simplified API” for EJB 3.0
- Critical examination of EJB complexity
  - “Data-mining” of common complaints, RFEs
  - Anti-patterns, etc.
- Focus on most common cases
- Ensure EJB 2.0/2.1 APIs are still supported
  - Simplifications / enhancements to improve use of these APIs as well

EJB 3.0 Direction

- Simplify developer experience
  - EJB's now Plain Old Java Objects (POJO)
  - Can use metadata annotations
    - XML descriptors are no longer necessary
    - Use defaults when possible
  - Unnecessary artifacts are optional
  - Simplify resource access using dependency injection
- Standardize persistence API for Java platform
  - Based on success of leading ORM solutions

EJB 3.0 Overview: POJOs

- Simplification of enterprise bean types
  - More closely resemble “POJOs”
- Elimination of requirement for EJB component interfaces
  - Use “POJIs” for session beans
  - No requirement for entity bean interfaces
- Elimination of requirement for Home interfaces
- Elimination of requirement for callback interfaces
  - Allow selective implementation of callback methods

EJB 3.0 Overview: Persistence

- Simplification of container-managed persistence
  - POJO / JavaBeans architecture approach
  - Support use of new()
  - Allow for testing outside the container
- Support for light-weight domain modelling, including
  - Inheritance and polymorphism
  - Object-relational mapping metadata
- Elimination of need for data transfer objects and related anti-patterns

EJB 2.1 Session Bean

```
import javax.ejb.EJBObject;
import java.rmi.RemoteException;
public interface Cart extends EJBObject
{
  public void add(String item) throws RemoteException;
  public Collection getItems() throws RemoteException;
  public void completeOrder() throws RemoteException;
}
```

```
import javax.ejb.*;
import java.rmi.RemoteException;
public interface CartHome extends EJBHome
{
  public Cart create() throws CreateException,
                              RemoteException;
}
```

ORACLE

EJB 2.1 Session Bean Class

```
import javax.ejb.*;
public class CartEJB implements SessionBean {
  protected Collection items = new ArrayList();
  public void add(String item) {
      items.add(item);
  }
  public Collection getItems() {
      return items;
  }
  public void completeOrder(){ .. }

  public void ejbCreate(){}
  public void ejbRemove(){}
  public void ejbActivate(){}
  public void ejbPassivate(){}
  public void setSessionContext(SessionContext context){}
}
```

EJB 2.1 Deployment Descriptor

```
<session>
   <display-name>Shopping Cart</display-name>
   <ejb-name>MyCart</ejb-name>
   <home>CartHome</home>
   <remote>Cart</remote>
   <ejb-class>CartEJB</ejb-class>
   <session-type>Stateful</session-type>
   <transaction-type>Container</transaction-type>
</session>
…
<assembly-descriptor>
   <container-transaction>
      <method> …
      </method>
      <trans-attribute>Required</trans-attribute>
   </container-transaction>
</assembly-descriptor>
```

Pain points

- Heavy-weight programmer view
- Bean class has no compile time dependency on business interfaces
- Requires methods that serve no purpose
- Extend a lot of interfaces and implementation classes
- Too many artifacts
  - Remote, RemoteHome, Local, LocalHome, WebService EndPoint, Bean & descriptors
- Complex XML deployment descriptors

EJB 3.0 Session Bean

```
public interface Cart {
   public void add(String item);
   public Collection getItems();
   public void completeOrder();
}
```

EJB 3.0 Session Bean

```
@Remote
public interface Cart {
   public void add(String item);
   public Collection getItems();
   public void completeOrder();
}
```

EJB 3.0 Session Bean class

```
import javax.ejb.*;


public class CartBean implements Cart {
    private ArrayList items;

    public void add(String item) {
        items.add(item);
    }
    public Collection getItems() {
        return items;
    }

    public void completeOrder() {...}
}
```

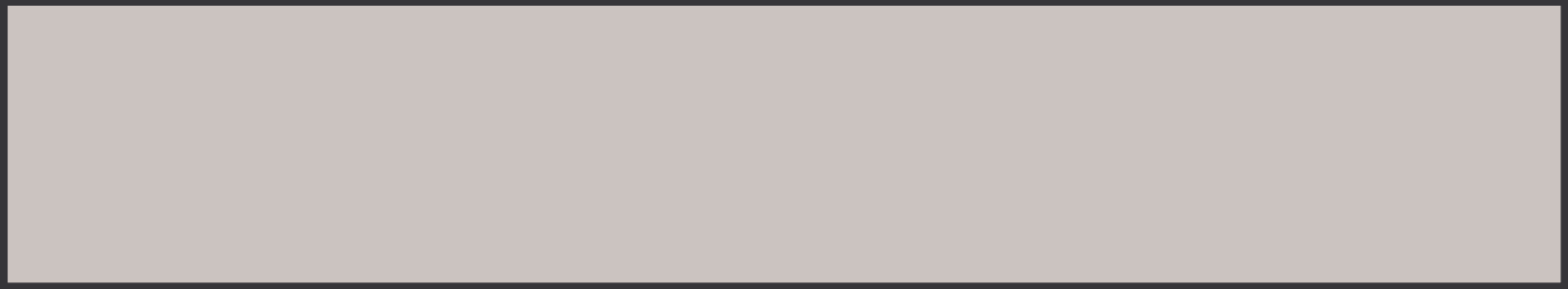
EJB 3.0 Session Bean class

```
import javax.ejb.*;

@Stateful
public class CartBean implements Cart {
    private ArrayList items;

    public void add(String item) {
        items.add(item);
    }
    public Collection getItems() {
        return items;
    }
    @Remove
    public void completeOrder() {...}
}
```

Deployment Descriptor



Significant Simplifications

- Eliminated requirement for Home Interface
  - Not needed for session beans
- Business interface is a POJI
  - Don't need to extend specific interface
  - Bean implements business interface
  - Bean can have more than one business interface
  - RemoteExceptions are removed from programmer and client view
- Eliminated requirement for unnecessary callback methods
  - Removed requirement to implement javax.ejb.SessionBean

Simplified Access to Bean's Environment

- Get JNDI APIs out of developer's view
- Declarative expression of dependencies in metadata
- Techniques/Mechanism
  - Container injection of resource entries, etc.
  - Simple programmatic lookup mechanisms
- Different usages, both have their place
  - Very simple; facilitates testability (especially setter injection techniques)
  - More flexible; dynamic

Injection Example

```
@Stateless public class MySessionBean {

  @Resource public DataSource customerDB;
  public void setDataSource(DataSource myDB) {
    customerDB = myDB;
  }

  public void myMethod (String myString){
     try {
           Connection conn =
                 customerDB.getConnection();
     ...
    } catch (Exception ex) {...}
  }
}
```

Persistence == POJO Entities

- Concrete classes (no longer abstract)
- No required interfaces
  - No required business interfaces
  - No required callback interfaces
- Supports new()
- getter/setter methods are not abstract
  - can contain logic (e.g., for validation, etc.)
- Usable (and testable) outside the EJB container

EJB 2.1 Entity Bean Interfaces

```
import javax.ejb.*;
public interface Customer extends EJBLocalObject
{
   public abstract String getName();
   public abstract void setName (String name);
   public abstract Account getAccount();
   public abstract void setAccount(Account acct);
}
```

```
import javax.ejb.*;
public interface CustomerHome extends EJBLocalHome
{
   public abstract Customer create(String name);
   public abstract Customer
                  findByPrimaryKey(String name);
}
```

EJB 2.1 Entity Bean class

```
import javax.ejb.*;
public abstract class CustomerBean
                          implements EntityBean {

   public abstract String getName();
   public abstract void setName(String name);
   public abstract Account getAccount();
   public abstract void setAccount(Account acct);}

   // plus life cycle methods…
```

EJB 2.1 Entity Bean class

```
    // Life cycle methods

    public String ejbCreate(String name)
         throws CreateException {…}
    public String ejbPostCreate(String name)
         throws CreateException {}
    public void ejbRemove() throws RemoveException {}
    public void ejbActivate() {}
    public void ejbPassivate() {}
    public void ejbLoad() {}
    public void ejbStore() {}
    public void setEntityContext(EntityContext ec) {}
    public void unsetEntityContext() {}
}
```

Developer Pain Points

- All the complexity of session beans and more
- Entity relationships are defined through complicated XML
- Bean class mixes business method and home method implementations
- Cannot be tested outside of the server
  - Abstract class with no testable state
  - Local interfaces can only be tested by servlets or session beans

EJB 3.0: EntityManager

- No Entity Bean Homes
- EntityManager serves as un-typed home or “session”
  - Provides lifecycle operations: create(), remove(), merge() etc.
  - Factory for Query objects
  - Can wrapper EntityManager to provided typed homes
- Container Managed
  - Integrated within an EJB 3.0 container
  - Allows for EM Injection and implicit JTA TX Support
- Application Managed
  - Application controls lifecycle of EM
  - Available within or outside container

EJB 3.0 Entity

```
@Entity public 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;
   }
}
```

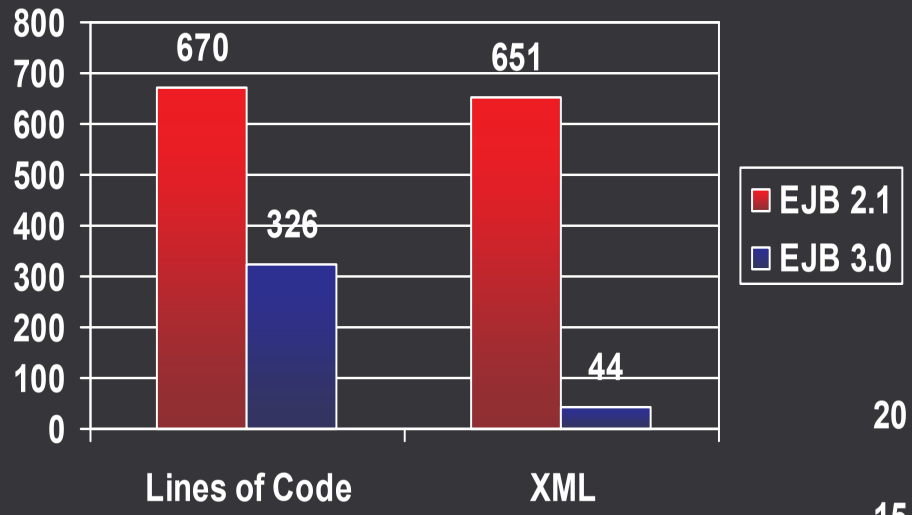
EJB 3.0 Entity

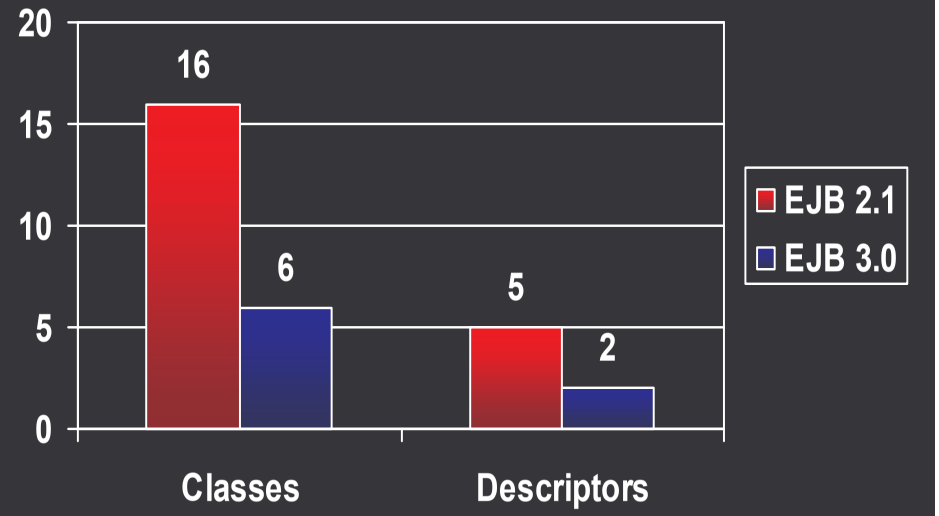
```
@Entity public class Customer {

   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;
   }
}
```

EJB 2.1 Versus 3.0: Simplifying Complexity





ORACLE

Flexibility and Performance

- Flexibility and performance are not being ignored for ease of development
  - O/R Mapping
    - Custom type mappings, inheritance …
  - EJB QL Enhancements
    - Sub queries, bulk operations, group-by …
  - Named queries, direct SQL, dynamic queries
  - Eager and lazy relationship fetching
  - Optimistic locking – version & timestamp

Oracle Support of EJB 3.0

- Oracle name co-spec lead on JSR 220 Expert Group
- Oracle to develop EJB 3.0 Persistence Reference Implementation
- Released EJB 3.0 preview in March 2005
- IDE Support for EJB 3.0 Persistence
  - Leading EJB 3.0 Eclipse project
  - JDeveloper (FREE!) 10.1.3 will include rich support
- Today Oracle TopLink supports both POJO and EJB 2.1 and 3.0
- Substantial EJB 3.0 functionality in 10.1.3 release

otn.oracle.com/ejb3

EJB 3.0 Summary

- Simpler to Build Components
  - Java Classes are EJBs
- Simpler to Lookup
  - Direct Methods; Dependency Injection
- Simpler to Map to RDBMS
  - Java Metadata for ORM; QL Improvements
- Simpler to Customize
  - Interceptor Methods
- Simpler to Deploy
  - Metadata Annotations
- Simpler to Test