



Application Development with Enterprise
JavaBeans™ Technology 2.0 Specification

Faiz Arni
InferData Corporation
President
farni@inferdata.com
www.inferdata.com

www.inferdata.com

Contents

- Module 1: Overview of EJB 2.0 Features
- Module 2: Local Interfaces
- Module 3: Developing EJB 2.0 CMP Entity Beans
- Module 4: EJB Query Language
- Module 5: Developing Message-driven Beans



Module 1:

Overview of EJB 2.0 Features

Overview

- Major new features introduced in EJB 2.0:
 - Local interfaces
 - Improved architecture for container-managed persistence
 - Container-managed relationships for CMP entity beans
 - EJB Query Language (EJB QL)
 - Message-driven beans

Local Interfaces

- In EJB 1.1, entity and session beans implement (indirectly) remote interfaces
 - Both `EJBHome` and `EJBObject` extend `java.rmi.Remote` interface
 - Marshal/unmarshal overhead occurs regardless of whether components are located on the same Virtual Machine (VM) or not
 - Parameters and return types are passed by value
- EJB 2.0 introduces the interfaces, `EJBLocalHome` and `EJBLocalObject`
 - Clients are collocated
 - No marshal/unmarshal overhead
 - Parameters are passed by reference

Improved Container-Managed Persistence (CMP)

- CMP fields are no longer defined in bean class
 - Abstract accessor methods for these fields are defined
 - Entity bean class is an abstract one
- CMP fields are described in deployment descriptor(DD)
- Based on DD and abstract accessor methods, container generates concrete bean class that extends the abstract bean class
- Containers are required to support both EJB 1.1 and 2.0 CMP models

Container-Managed Relationships

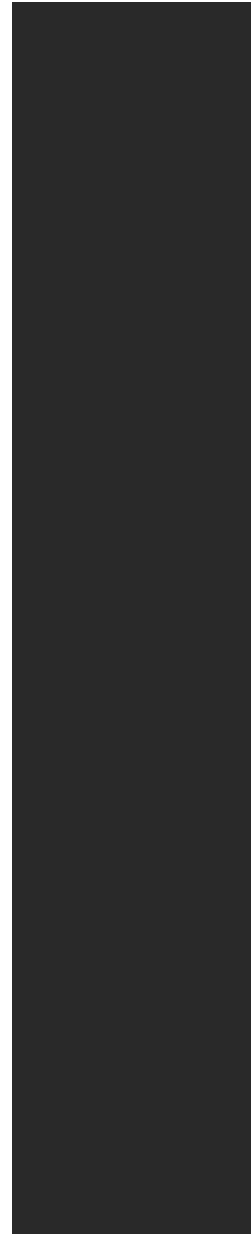
- Entity beans are related to other entity beans modelling the standard database table relationships
 - Eg. A customer may have several addresses
- EJB 2.0 lets container manage the relationships for you -- container-managed relationships (CMR)
- CMR supports:
 - One-to-one
 - One-to-many
 - Many-to-one
 - Many-to-many

EJB Query Language (EJB QL)

- Problems with EJB 1.1 in defining CMP finder methods
 - Vendor specific query language
 - Non-portable
- EJB QL standardizes the way to write CMP *query methods*:
 - Finder methods
 - Select methods
- Based on SQL 92
 - Portable across all EJB 2.0 compliant containers

Message-driven Beans (MDB)

- Asynchronous messaging allows decoupling of senders from receivers
- Message-driven Beans process asynchronous messages delivered by ***any*** message producer
 - Serve as router processes that operate on inbound messages
- Use message-driven beans to
 - Integrate an EJB-based system with a legacy system
 - Enable business-to-business interactions
 - Implement load-balancing functionality



Module 2:

Local Interfaces

Why Local Interface

- In EJB 1.1:
 - Remote marshal/unmarshal overhead occurs regardless of whether components are located on the same JVM or not
 - Parameters are passed by value
- In EJB 2.0:
 - Corresponding to `EJBHome` there is `EJBLocalHome`
 - Corresponding to `EJBObject` there is `EJBLocalObject`
 - `EJBLocalHome` and `EJBLocalObject` support clients collocated in the same EJB container
 - No marshal/unmarshal overhead
 - Parameters are passed by reference

EJBLocalHome

- An enterprise Bean's local home interface serves the same purpose as remote home interface
- All enterprise Beans' local home interfaces must extend the EJBLocalHome interface

```
public interface EJBLocalHome {  
    // Remove an EJB object identified by its primary key.  
    void remove(java.lang.Object primaryKey)  
        throws RemoveException, EJBException  
}
```

EJBLocalObject

- An enterprise Bean's local interface defines the business methods callable by local clients.
- All enterprise Beans' local interface must extend the EJBLocalObject interface

```
public interface EJBLocalObject {  
    // Obtain the enterprise Bean's local home interface  
    public EJBLocalHome getEJBLocalHome() throws EJBException  
    // Obtain the primary key of the entity EJB local object  
    public java.lang.Object getPrimaryKey() throws EJBException  
    // Remove the EJB local object  
    public void remove() throws RemoveException, EJBException  
    // Test if it is identical to the invoked EJB local object  
    public boolean isIdentical(EJBLocalObject obj) throws  
EJBException  
}
```

Local Interface Characteristics

- A local interface *must be used in the same JVM* the EJB is deployed in
- Parameters and return values on a local interface are passed by reference
 - Bean Provider must be aware of the potential sharing (by caller and callee) of objects passed through the local interface
- Methods on local interface do not throw `java.rmi.RemoteException`
- Home object returned from JNDI lookup needs not to be narrowed; a normal Java cast is adequate

Local and Remote Interface

- The Bean Provider *must* provide
 - A remote home interface and a remote interface, *or*
 - A local home interface and a local interface
- The Bean Provider *may* provide
 - A remote home interface,
 - A remote interface,
 - A local home interface, *and*
 - A local interface
- Other combinations are not supported
- Usually an EJB provides either a remote or local interface, not both



Module 3:

Developing EJB 2.0 CMP Entity Beans

Problems of EJB 1.1 CMP Model

- Difficult to model coarse-grained entity beans
 - Usually requires use of dependent objects to model complex state or relationships
 - As another entity bean
 - Poor performance
 - Simple Java class (dependent value class)
 - Difficult to save the state to database
- Lack of portable way to specify finder queries
 - Each vendor has its proprietary way of defining finder methods

CMP 2.0 Solves the Problems

- Local interfaces allow fine-grained entity beans to be used as dependent objects
 - Both efficiency and persistence issues are solved
- EJB QL provides portable way to write finder methods

Entity Bean Class

- The entity bean class must be defined as an abstract class
 - The container generates the implementation class that is used at runtime
- The CMP and CMR fields must *not be defined* in the entity bean class
- The local interface types of the entity bean and of related entity beans must not be exposed through the remote interface of the entity bean

Abstract Accessor Methods

- The entity bean class must define the accessor methods for the CMP and CMR fields as *get* and *set* methods
- The accessor methods must
 - Be `public`
 - Be `abstract`
 - Bear the name of the CMP field or CMR field
- The implementation of the accessor methods is supplied by the Container
- The accessor methods for the CMR fields must not be exposed in the remote interface of an entity bean

Using Accessor in Entity Bean Class

- The entity bean class uses the accessor methods to access CMP and CMR fields

```
abstract public class CustomerBean implements EntityBean {
    // container managed fields
    abstract public String getCustomerId();
    abstract public void setCustomerId(String id);
    abstract public String getName();
    abstract public void setName(String name);
    abstract public int getAge();
    abstract public void setAge(int val);
    public String ejbCreate(String name, int age) throws CreateException {
        String uid = this.getUniqueIdForCustomer();
        setCustomerId(uid);
        setName(name);
        setAge(age);
        return null;
    }
    ...
}
```

Allowed CMP Field Types

- Legal CMP field types are:
 - Java primitive types: int, long, float, etc.
 - Java serializable types
- An entity bean local interface type (or a collection of such) cannot be the type of a CMP field

Abstract Persistence Schema

- A set of XML elements in the deployment descriptor defines an *abstract persistence schema*
 - <abstract-schema-name>
 - <cmp-field>
 - <relationships>
 - <cmr-field>, etc.
- It defines the characteristics of the CMP and CMR fields

CMP Field Deployment Descriptor

- The CMP fields must be specified in the deployment descriptor using the `<cmp-field>` elements
- Each CMP 2.X entity bean must have a unique `<abstract-schema-name>` in a DD file

```
<ejb-jar>
...
<enterprise-beans>
  <entity>
    ...
    <cmp-version>2.x</cmp-version>
    <abstract-schema-name>Customer</abstract-schema-name>
    <cmp-field>
      <description>no description</description>
      <field-name>age</field-name>
    </cmp-field>
  ...
```

Container-Managed Relationships

- CMR are defined in terms of the local interfaces of the related beans.
- An entity bean that does not have a local interface can have only unidirectional relationships from itself to other entity beans
 - The lack of a local interface prevents other entity beans from having a relationship to it
- Relationships may be either bi-directional or unidirectional

Allowed CMR Field Types

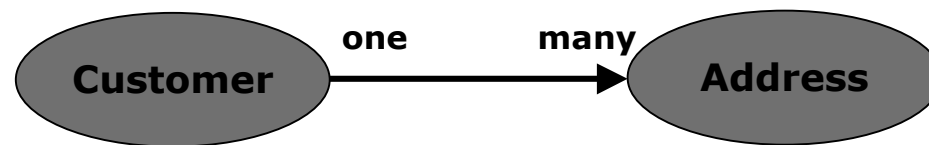
- Legal CMR field types are:
 - An entity bean local interface type
 - A collection of local entity bean interface types

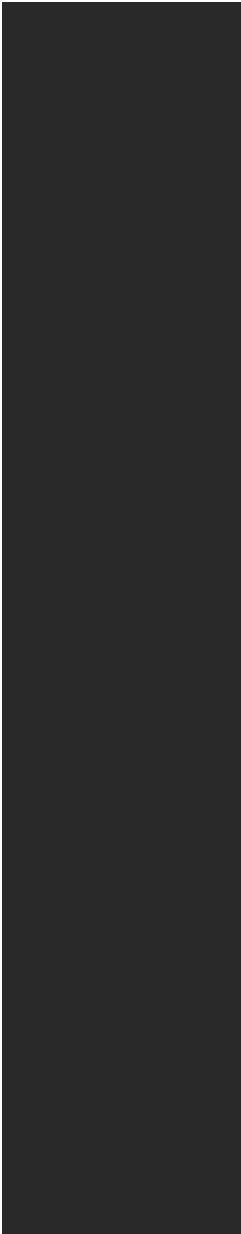
Defining Relationships

- Relationships are expressed using `CMR` fields in the deployment descriptor
 - Use the object type of the entity bean to represent the "one" side
 - Use Java Collection to represent the "many" side
 - The implementation of the collection is supplied by the Container

CMP 2.0 Example

- Example models the scenario where a customer may have several addresses
 - A customer table
 - An address table with customerId as foreign key into the customer table
 - One-to-many Relationship: a customer may have many addresses





Module 4:

EJB Query Language

What is EJB Query Language

- EJB Query Language (EJB QL) is for defining query methods in a vendor independent way
 - Before EJB 2.0, each vendor has its own way
- Based on SQL-92 syntax
 - Tailored to work with the abstract persistence schema of CMP 2.0
- Can be used in two different ways
 - Write queries for finder methods
 - Writing queries for abstract ejbSelectXXX() methods

EJB QL Translation

- EJB QL queries are defined in terms of the abstract persistence schema of entity beans
 - Not in terms of the underlying data store
- At run time, query methods typically execute in the native language of the underlying data store.
 - A container that uses a relational database for persistence might translate EJB QL statements into standard SQL 92
 - An object-database container might translate the same EJB QL statements into an object query language.

Declaring EJB QL Queries

- EJB QL statements are declared in `<query>` elements of entity bean's deployment descriptor
- For example:

```
<query>
  <description></description>
  <query-method>
    <method-name>findByName</method-name>
    <method-params>
      <method-param>java.lang.String</method-param>
    </method-params>
  </query-method>
  <ejb-ql>
    select distinct object(c) from Customer c where c.name = ?1
  </ejb-ql>
</query>
```

EJB QL Queries

- EJB QL is expressed in terms of the abstract persistence schema:
 - Abstract schema name
 - CMP fields
 - CMR fields
- An EJB QL Query must contain a `SELECT` and a `FROM` clause; may contain a `WHERE` clause

EJB QL ::= select_clause from_clause [where_clause]

- Input parameters are referenced as ?1, ?2, etc. which are mapped to the `<method-param>` elements in `<method-params>`

Abstract Schema Types and Query Domains

- There is a one-to-one mapping between *abstract schema types* and entity bean homes
- Abstract schema names, specified by `<abstract-schema-name>` elements in the DD, are used to denote abstract schema types in EJB QL
- The *domain of an EJB QL query* consists of the abstract schema types of all CMP entity beans in the same DD
- Allowable types in EJB QL are:
 - Abstract schema types of entity beans
 - Types of CMP fields

Select Clause

- SELECT clause specifies the value of a query result
- Allowed result type:

- Abstract schema types of entity beans

➤ For example, to find all orders:

```
SELECT OBJECT(o) from ORDER o
```

- CMP field type [only for ejbSelect methods]

➤ For example, to find all the cities from shipping addresses of all orders:

```
SELECT o.shipping_address.city FROM Order o
```

FROM Clause

- The FROM clause defines the domain of the query by declaring identification variables
- The domain of the query may be constrained by path expressions
- *Identification variables* designate instances of a particular entity bean abstract schema type
- Example:
 - Find orders with quantities larger than John Doe's:

```
SELECT DISTINCT OBJECT(o1)
FROM Order o1, Order o2
WHERE o1.quantity > o2.quantity AND o2.customer.lastname='Doe'
      AND o2.customer.firstname='John'
```

Where Clause

- The WHERE clause restricts the result of a query by defining a conditional expression
- Conditional expressions consist of:
 - Comparison operations:
 - =, >, >=, <=, <>
 - BETWEEN, IN, LIKE, IS [NOT] NULL, IS [NOT] EMPTY, [NOT] MEMBER [OF]
 - Logical operations: NOT, AND, OR
 - Path expressions that evaluate to boolean values
 - Boolean literals



Module 5:

Developing Message-driven Beans

Message-driven Beans

- A **message-driven bean** (MDB) is
 - An asynchronous message consumer
 - A `MessageListener` object on a JMS destination
 - Invoked by the container as a result of the arrival of a JMS message
- A message-driven bean has neither a home nor a remote interface
- A message-driven bean has no conversational state
- A message-driven bean is anonymous

Association of MDB With JMS Destination

- Bean Provider uses `message-driven-destination` deployment descriptor element to advise Deployer which JMS destination this MDB is to be associated with.
- An MDB is associated with a JMS Destination (`Queue` Or `Topic`) when the bean is deployed.

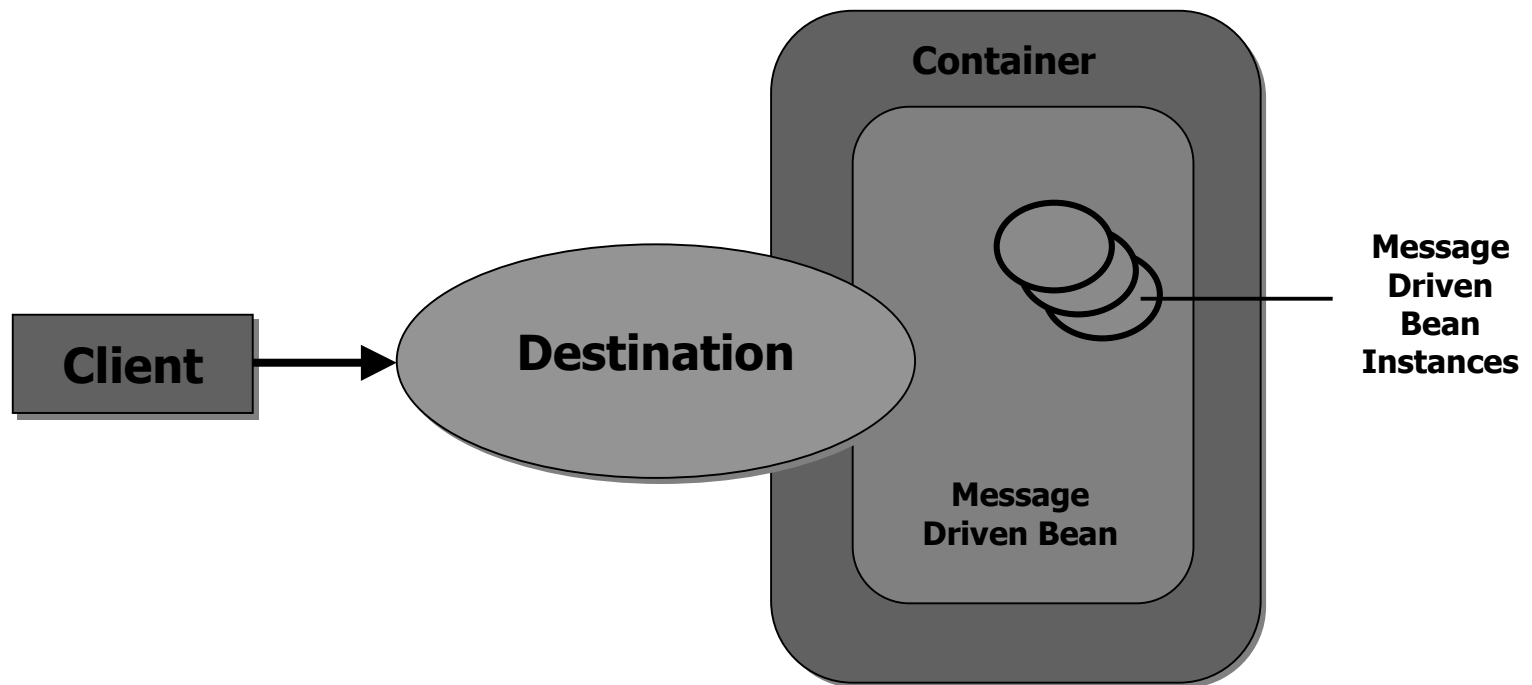
Client View of an MDB

- To a client, an MDB is simply a JMS message consumer (`MessageListener` for a JMS Destination).
- Client sends message to the JMS Destination.
- Clients locate a JMS Destination (for which an MDB is a listener) by using JNDI:

```
Context initialContext = new InitialContext();  
Queue stockOrder = (javax.jms.Queue)initialContext.  
    lookup("java:com/env/jms/stockOrder");
```

Client View of an MDB

Client view of MDBs deployed in a container:



Message-driven Bean Class

- All message-driven beans must implement the `MessageDrivenBean` and the `MessageListener` interfaces:

```
package javax.ejb;

public interface MessageDrivenBean {
    public void ejbRemove();
    public void setMessageDrivenContext(MessageDrivenContext c);
}

package javax.jms;

public interface MessageListener {
    public onMessage(Message msg);
}
```

- In addition, they must also implement an `ejbCreate()` method.

onMessage() Method

- The `onMessage()` method is called by the bean's container when a message has arrived for the bean to service.
- The `onMessage()` method contains the business logic that handles the processing of the message.
- The `onMessage()` method has a single argument, the incoming message.

Message-driven Bean Lifecycle

- Container invokes `newInstance()` to create a new MDB instance
- Container calls `setMessageDrivenContext()` to set the context
- Container calls `ejbCreate()`
- The MDB instance is ready to be delivered messages (`onMessage()` method is called)
- When the container no longer needs this MDB instance (e.g. to reduce pool size), `ejbRemove()` is called to end the life of this MDB instance.

Summary

- EJB 2.0 Features
 - Local Interfaces for efficient implementation of non-remote objects
 - Specification of entity relationships in a portable and elegant manner to model business database schemas
 - EJB Query Language for specifying *finder* methods and *select* methods in a portable and standard way
 - Modeling asynchronous processing using Message-driven Beans

Resources

- Read our white papers at:
 - www.inferdata.com/whitepapers
- Coming soon:
 - Advanced J2EE Design Patterns
 - From Rational Rose™ Models to Enterprise Beans
 - Model-Driven Architectures for the Enterprise
 - Designing and Implementing High Performance E-Business Systems