Some EJB Best Practices

Kyle Brown
Senior Java Consultant
IBM WebSphere Services
RTP, NC
brownkyl@us.ibm.com

Overview

- Enterprise Java Beans Definitions
- When to use EJB's
- EJB Architectures
- Best Practices

Enterprise Java Beans

- A component model for developing and deploying object-oriented, multitier Java applications
- Based on three foundational technologies
  - OMG's CORBA
  - RMI (Java Remote Message Invocation)
  - X/Open XA Transaction architecture

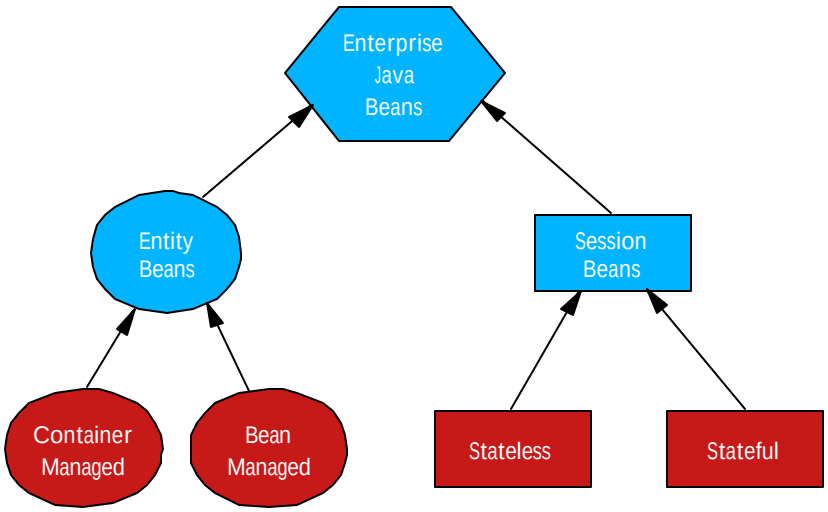
So what's an EJB?

- An EJB is a distributed, transactional, (possibly) persistent software component
- Programmer provides a specification and implementation for its business methods
- Transaction attributes and persistence can be provided at Deployment time

Bean Types

- Session beans
  - typically represent function
  - belong to a single client for a period of one or more method calls
  - created and destroyed by a client (transient)
- Entity beans
  - shared by multiple clients
  - typically represent persistent data

EJB Types



Stateless vs. Stateful Session Beans

- Stateless
  - may not possess client-specific internal state
  - can be pooled to service multiple clients
- Stateful
  - may possess client-specific internal state
  - one per client

Entity Bean persistence

- Container-managed
  - Container is responsible for saving state
  - Persistence is through generated code
  - persistence independent of data source
- Bean-managed
  - Bean is responsible for saving its own state
  - Developer is responsible for persistence
  - Less adaptable; persistence is hard-coded
  - More flexible

Qualification Critera

- The following questions can help you decide if you need EJB's
- If you answer "yes" to any of the questions, you may benefit from EJB's

Standards

- Do you need a standards-based architecture?
  - Multiple Vendor support
  - Training issues
- EJB's are standards-based.
  - It is easier to cross-train developers to a standard than a proprietary base.
  - In a multi-vendor environment you can leverage common code and experience

- At the same time, each vendor goes beyond the EJB specification in small ways. For instance, VisualAge for Java and WebSphere provide some features for CMP that go beyond the specification. WebLogic likewise allows for setting some configuration options at runtime that go beyond the specification.
- However the bulk of your code (90%+) should be vendor-independent. It is your choice whether or not to use vendor-specific features.
- One thing to keep in mind is that EJB IS the standard for TP systems in the future. All IBM TP systems will either migrate to EJB compliance or merge into EJB compliant systems (including CICS...)

Transactional Support

- Do you need to access multiple datasources with transactional support?
  - This is the argument for traditional TP systems
  - EJB's provide a standard, vendor-independent architecture for building TP systems
  - Currently limited 2PC support but getting better...

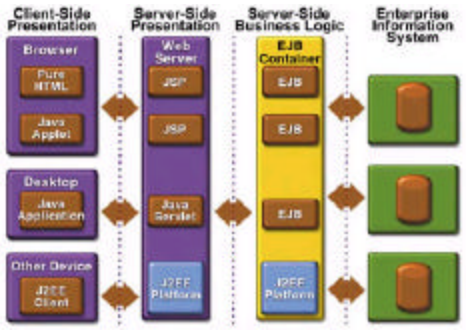
- For instance, we Currently do not support 2PC for Oracle because of problems with Oracle's XA JDBC Driver.
- However, this will be fixed in the future. We have also announced plans to support transactional 2-PC with MQ Series through our MQ Series JMS classes.

Client Support

- Do you need to support multiple client types?
  - HTTP, Desktop Application, WAP, etc.
- EJB's (through RMI/IIOP) allow multiple client types to access the same back-end logic
- This feature is a key to the J2EE application architecture

J2EE Architecture

- Many client types can access EJB's
  - Servlets & JSP's
- Clients can go through RMI/IIOP
  - May use other protocols (SOAP/XML)



Security Support

- Do you need method-level security on your objects?
- EJB's allow you to specify how users and groups can be allowed or denied access to both EJB types and individual EJB methods.
- WebSphere uses standard Security API's -- LDAP, SSL, etc.

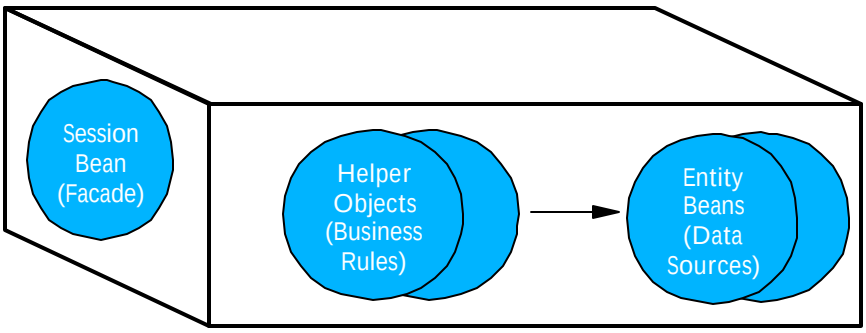
- Conversely, one of the things that EJB's do *not* allow you to easily is object-level security. It can be done in the WebSphere environment since we allow programmatic access to authentication information and other user information that is stored in LDAP, but it will require custom coding.
- If this is one of your key requirements, you may need to carefully evaluate the amount of work necessary to achieve it.
- Another thing that is not provided in EJB's is a connection between the "logged-in" user and the database-level user. If you require that each user have their own unique database id and connection, then EJB's may not be the best option for you.

EJB Program Architectures

- Clients should not be aware of
  - the complexity of your entity model
  - the way in which you manage data
- Use session beans as *Facades*
  - Minimize distribution "cross-section"
  - Hide Entity beans or other persistence mechanisms
  - Java Beans (Helper Objects) implement business logic

- Facade is a design pattern from Gamma, et. al. A Facade is an object that hides the complex relationships between other objects.
- By "Java Bean" I really just mean any serializable Java Object.

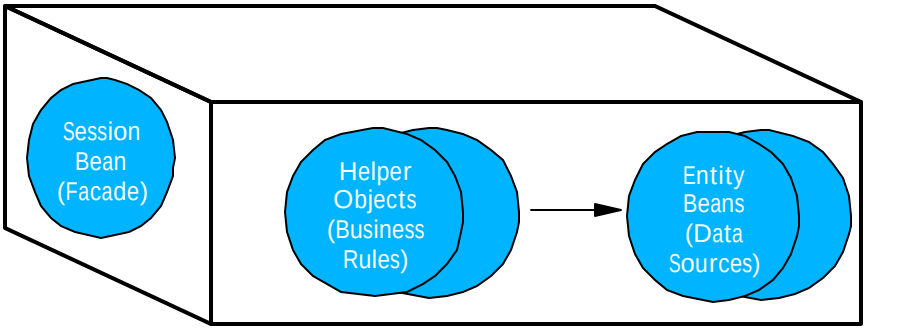
EJB Architecture



Session Bean API is the client's view of the system

Helper Objects "inside the box"

- Application-specific domain objects
  - implement business rules
  - perform validation and integration of disparate data

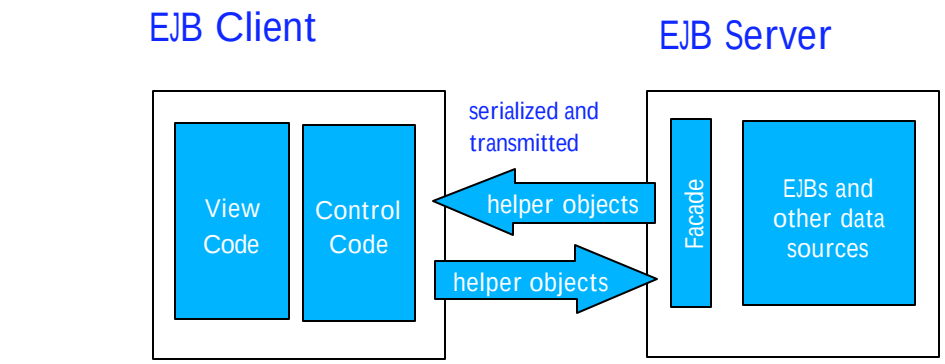


IBM

- Helper Objects are sometimes called Dependent Objects. This terminology is used in the EJB 1.1 specification and is extensively used in the EJB 2.0 specification, where it takes on a slightly different (but mostly compatible) meaning.
- Validation should be performed as quickly as possible by Helper objects rather than by the Entity beans themselves. Validating early in the helpers reduces the number of bean references that participate in the transaction (and are therefore locked) in the case of a validation failure, which must be rolled back. This will reduce contention in your application and speed things up.

Helper Objects "outside the box"

- Can be a communication vehicle from facade to client
  - Copied by value; must be declared Serializable



Control code modifies the helpers according to the user's actions.

- Any object that is serializable can be returned as the value of a Session bean's method. This object will be serialized and copied across the network to the requesting client. Likewise modified helper objects can be passed back to the session bean in the same way.
- Remember that accesses to an EJB are network calls. This can adversely affect performance when you have a large number of fine-grained objects that you access from the client. Using helper objects can reduce this network traffic. The client can make several changes at once to a single helper object and then send that helper object to the facade as one network call instead of several -- one for each change.
- Of course, WebSphere can be optimized to avoid overhead like this when the client and EJB are in the same JVM.

Pros and Cons of Session Facades

- Advantages of session management
  - Entity beans are general-purpose data sources
  - Session beans can tie together multiple data sources
  - Additional transaction management through SessionSynchronization
- Disadvantages of session facades
  - The total model is more complex

Minimize Entity Bean Coupling

- Avoid the temptation to build create() and finder methods in your Entity Beans that use Helper objects
- This ties the Entities to specific Helpers
  - Creates a class dependency nightmare
  - Makes deployment difficult
  - Makes Reuse of the Entity Beans challenging

Common scenarios

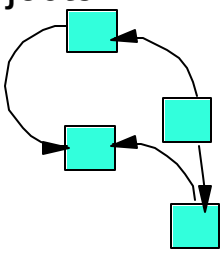
- The following set of scenarios will help you understand when to apply different EJB types
- We will set up some problems and then illustrate why each problem is best solved with a particular EJB design pattern

Basic Assumptions

- Assume a standard architecture of Session Beans presenting facades to clients
  - The Session beans may wrap Entity beans, other Session beans, or Java classes
- Should **never** allow a client direct access to an Entity bean
  - Efficiency (each call is a **new** transaction)
  - Logical consistency (one write succeeds, another fails)

Complex Relationships

- Often you find a set of persistent objects with complex relationships
  - 1-N and 1-1 relationships
  - Inheritance
  - Object-relational mismatches



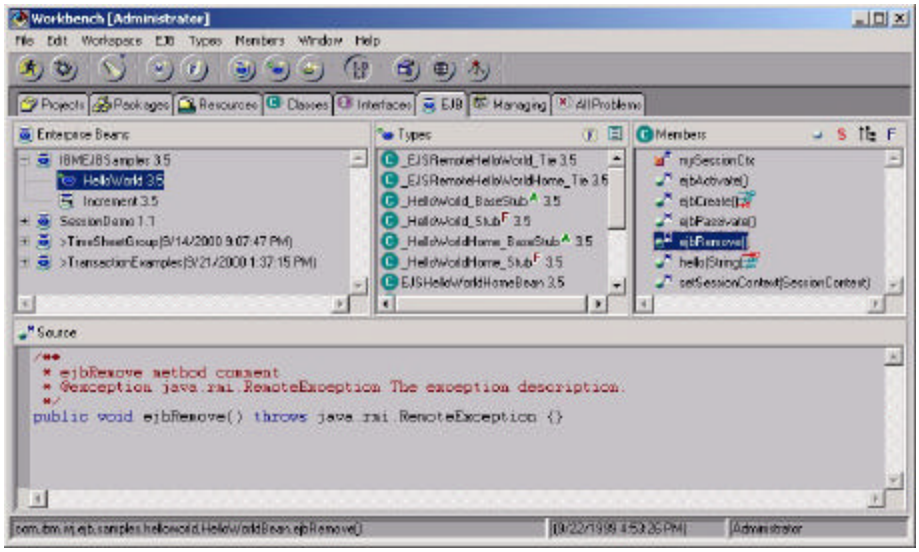
- When a user must navigate a tree or graph structure then often CMP EJB's are the best solution.

CMP advantages for Graphs

- WebSphere & VAJ together support CMP
  - Efficient (ex: 1 SQL call per finder vs. N+1)
  - Handles many cases without custom SQL coding
- EJB's are created and mapped to a DB schema in VAJ, tested in VAJ, exported to WebSphere for execution
- Easy to change when requirements change

- Note that there are a few cases where graph navigation might be better done in another way. This will be covered later.

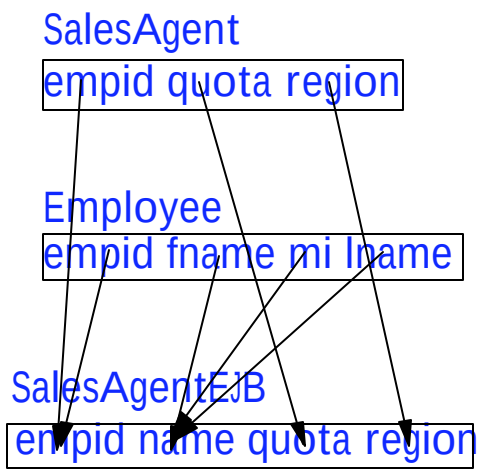
The EJB Page



- Things to notice on this page:
- (1) At the upper left is a listing of the EJB Groups, which is a tree view showing the EJB's contained within the EJB groups.
- (2) The middle pane shows the classes and interfaces associated with a particular EJB selected in the first pane. The right-hand icon toggles between a field view (useful only for Entity beans) and a type view. The left-hand icon toggles between seeing only the major classes and interfaces (Home, Remote, Bean) and all generated classes.
- (3) The leftmost page shows the methods that correspond to a particular type selected in the middle pane. The icon to the right of some methods show that these methods have been "promoted" to the Remote or Home Interface.
- (4) The lower pane shows the code of the class definition or method selected.

Primary Key Joins

- VisualAge supports joins on Primary keys
  - Create one "primary" table
  - Select N secondary "tables"
- Can select attributes from each



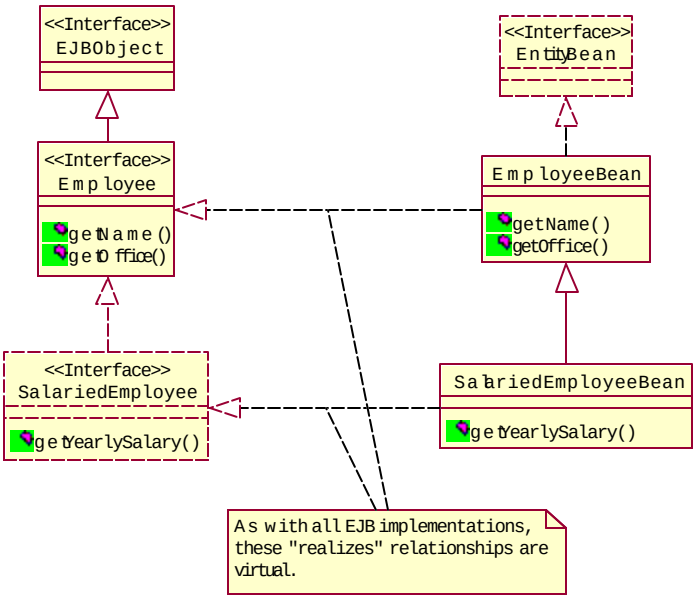
EJB Inheritance

- Remote interfaces can inherit from other remote interfaces
  - Permit extension of component services
  - Client calls can exploit polymorphism
- Bean implementation classes can inherit from other implementation classes
  - Code reuse

There are some common-sense limitations to this:

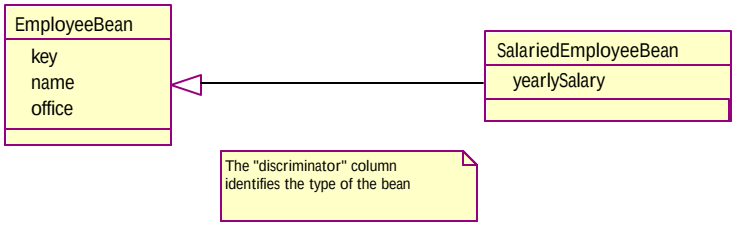
(1) Session beans cannot inherit from Entity Beans and vice versa.
(2) A Stateless session bean cannot inherit from a stateful session bean and vice-versa

Allowable Inheritance



Single-table Inheritance

key	discriminator	name	office	yearlySalary
101	EmployeeBean	Fred	B105	null
202	SalariedEmplyee	Bob	C238	40000

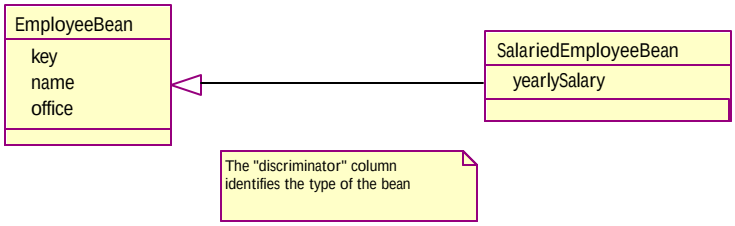


All instances in inheritance hierarchy stored in a single Table
Attributes not found in all instances must be nullable
Need a column dedicated to identifying actual type (discriminator).

Root-leaf Inheritance

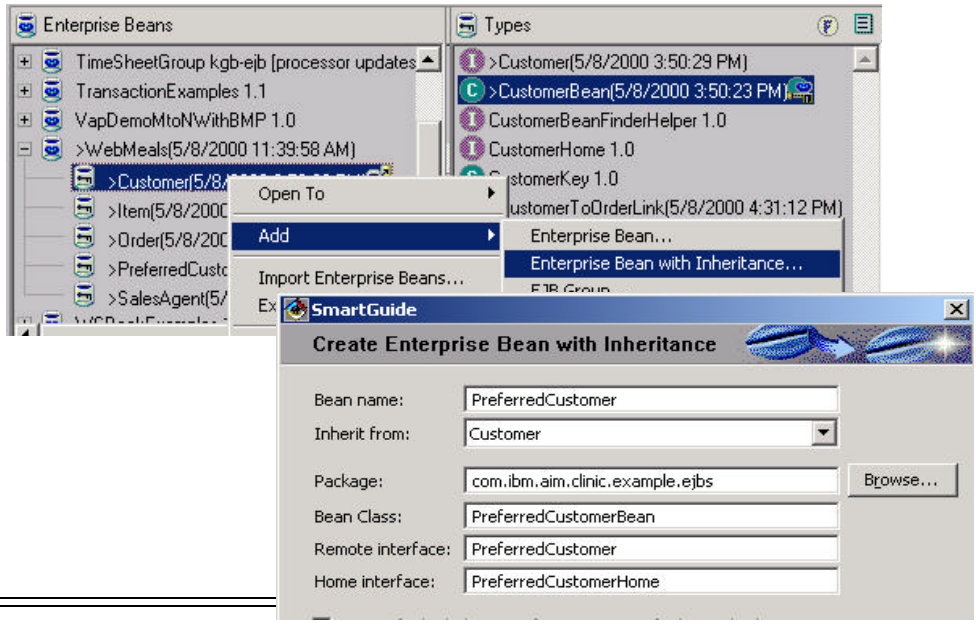
key	discriminator	name	office
101	EmployeeBean	Fred	B105
202	SalariedEmployee	Bob	C238

key	yearlySalary
202	40000

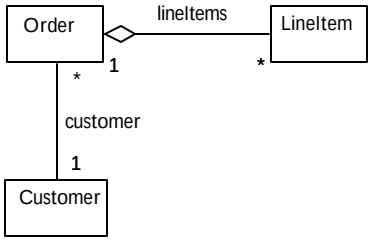


- This mapping save db space by eliminating nullable attributes.
- Object creation requires table joins to retreive all attributes.
- Still requires a discriminator column.
- Leaf tables key -- is a foreign key to root table.

VAJ Inheritance



Kinds of Object Relationships



- 0..1-0..1 relationships
- 0..1-N relationships
- M-N relationships

Order

orderid	fname	phone	addressFK
1011	Bob	212-3999	2023

Customer

phone	fname	mi	lname
212-3999	Bob	R	Stevens

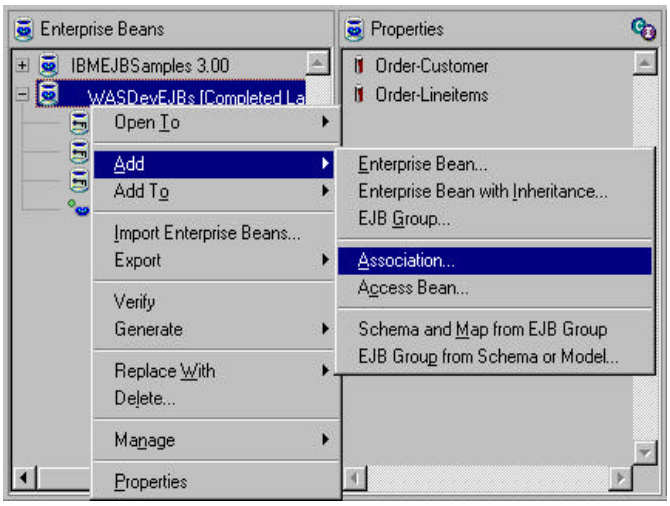
LineItem

itemid	description	quantity	orderFK
4027	TV Dinner	8	1011

- This slide shows different kinds of object relatonships. Suppose you have a Java class called "Order". An "Order" has two instance variables -- an "customer" that holds an instance of an Customer class and a "lineItems" variable that holds a collection of "LineItem" objects.
- The three tables below show how you might implement these relationships in a relational database. the N-1 relationship between Order and Customer is implemented by a foreign key in the Person table. The 1-N relationship between Order and LineItem is implemented as a foreign key in the LineItem table (a backpointer).
- M-N relationships (not shown) are usually implemented through a third, "relationship" table that includes foreign keys to both parties in the M-N relationship.

Setting Associations

- Associations are defined as special properties
  - Not defined as CMP fields
  - Defined after both EJBs in association are defined
  - Can be defined on either EJB in association



Complex SQL

- CMP Entities in WebSphere Advanced cannot handle arbitrary SQL
  - Joins (other than on shared primary keys)
  - Views
- If you need to handle these, you must use another solution
  - BMP
  - "Session wrapped" persistence framework

Relational Joins

- CMP does not handle relational joins well
  - Sometimes an object structure is naturally mapped to a join
  - Joins are often needed for efficiency
- In this case, BMP is the best solution **today** that is available

BMP's and Dependent Objects

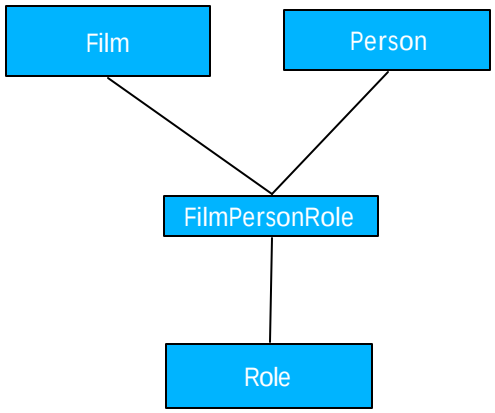
- You usually can't map the result of a join to a CMP EJB
  - Each CMP EJB corresponds to a single row in a table
  - No way of tying multiple EJB's together
- Instead use BMP with "Dependent objects" or "Helpers"
  - Java Beans (Serializable Java objects)
  - Persistence will be managed by a BMP bean or by outside persistence solutions

Session-Wrapped Persistence Layer

- A solution we've seen a few times is to use persistent JavaBeans (e.g VisualAge PB)
  - Build a "Transaction Wrapper" Stateful Session EJB that implements SessionSynchronization
  - Tie the PB and EJB transactions together
  - Allows you to transactionally tie together PB and CMP Entity objects
  - Provides distribution to PB

The N-ary relationship problem

- Often you encounter N-ary relationships
  - Could implement with VA-J CMP
  - Poor performance with this solution



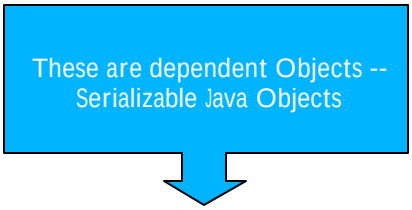
3N + 1 SQL Statements!

- The example here is "Ron Howard was an Actor in American Graffiti. Ron Howard was the Director of Cocoon"
- The first SQL statement is by the finder of the FilmPersonRole bean, which returns back all of the appropriate FilmPersonRoles.
- However, you must traverse each single-valued relationship individually. This results in a findByPrimaryKey() on each object, which is another SQL statement. For N FilmPersonRoles, with 3 relations, that makes a total of 3N + 1

Solving N-ary with BMP

- Can write a BMP that
  - Represents the FilmPersonRole Association
  - Returns *Dependent Objects (e.g. Film, Role, Person)*
  - A single SQL Statement in the ejbFindBy...() to retrieve keys (FilmCrewRole)
  - A SQL Statement in each ejbLoad() that does a 4-way join
- Total of N + 1 SQL Calls

More on the BMP Solution




```
public class FilmPersonRoleBean implements EntityBean {
    ...
      private Film film;
      private Person person;
      private Role role;
    ...
}
```

BMP EJB Finder method

```
public java.util.Enumeration ejbFindByFilm(int argFilmId) throws java.rmi.RemoteException, javax.ejb.FinderException {
        java.util.Vector results = new java.util.Vector();
        Connection jdbcConn = getConnection(); // get from the connection pool
        try {
                String sqlString = "SELECT filmId, personId, roleId FROM KBROWN.FilmPersonRole WHERE (filmId = ?)";
                PreparedStatement sqlStatement = jdbcConn.prepareStatement(sqlString);
                sqlStatement.setInt(1, argFilmId);
                ResultSet sqlResults = sqlStatement.executeQuery();
                while (sqlResults.next()) {
                        FilmPersonRoleKey newKey = new FilmPersonRoleKey();
                        newKey.filmId = sqlResults.getInt(1);
                        newKey.personId = sqlResults.getInt(2);
                        newKey.roleId = sqlResults.getInt(3);
                        results.addElement(newKey);
                }
        } catch (Exception e) { // DB error
                throw new FinderException("Database Exception " + e + "caught");
        } finally { try {jdbcConn.close()} catch (Exception e1) {}; }
        return results.elements();
}
```

BMP EjbLoad Method

```
public void ejbLoad() throws java.rmi.RemoteException {
        //_initLinks();
        boolean wasFound = false;
        boolean foundMultiples = false;
        FilmPersonRoleKey key = (FilmPersonRoleKey) getEntityContext().getPrimaryKey();
        Connection jdbcConn = getConnection();
        try {
                // SELECT from database
                String sqlString = "SELECT a.filmId, a.personId, a.roleId, b.title, c.name, d.name FROM KBROWN.FilmPersonRole a,
          KBROWN.Film b, KBROWN.Person c, KBROWN.Role d WHERE (a.filmId = ? AND a.personId = ? AND a.roleId = ?  AND
          a.filmId = b.id AND a.personId = c.id  AND a.roleId = d.id)";
                PreparedStatement sqlStatement = jdbcConn.prepareStatement(sqlString);
                sqlStatement.setInt(1, key.filmId);
                sqlStatement.setInt(2, key.personId);
                sqlStatement.setInt(3, key.roleId);

                // See the next page for the execution of the statement

        } catch (Exception e) { // DB error
                throw new RemoteException("Database Exception " + e + "caught in ejbLoad()");
        }
        if (wasFound && !foundMultiples) {
                return;
        } else {
                throw new RemoteException("Multiple rows found for unique key in ejbLoad().");
        }
}
```

Execution of the Query

```
// Execute query
ResultSet sqlResults = sqlStatement.executeQuery();
// Advance cursor (there should be only one item)
// wasFound will be true if there is one
wasFound = sqlResults.next();
      if (wasFound) {
         // If the Join ran correctly, then set the internal variables of the dependent objects
         film = new Film();
         film.setId(sqlResults.getInt(1));
         film.setTitle(sqlResults.getString(4));
         person = new Person();
         person.setId(sqlResults.getInt(2));
         person.setName(sqlResults.getString(5));
         role = new Role();
         role.setId(sqlResults.getInt(3));
         role.setName(sqlResults.getString(6));
}

// foundMultiples will be true if more than one is found.
foundMultiples = sqlResults.next();
```

Connection pooling

- When building BMP beans (or Sessions that do DB queries) ALWAYS use Connection pooling
- WebSphere connection pooling is the only way for the Txn Mgr to commit() your updates in the right place
- Uses the standard JDBC 2.0 Connection pooling classes (javax.sql.DataSource)

Example getConnection()

```
public Connection getConnection() throws SQLException, RemoteException {
    // ds can be a transient variable in this EJB or a Singleton
      if (ds == null) {
            try {
                  Properties props = new Properties();
                  props.put(Context.INITIAL_CONTEXT_FACTORY,
                "com.ibm.ejs.ns.jndi.CNInitialContextFactory");
                  props.put(Context.PROVIDER_URL, "iiop://localhost:900/");
                  // fill in the properties
                  InitialContext initContext = new InitialContext(props);
                  ds = (DataSource) initContext.lookup(DATASOURCE_NAME);
                  if (ds == null)
                        throw new RemoteException("Could not initialize DataSource");
            } catch (NamingException e) {
                  throw new RemoteException("Could not initialize DataSource " + e);

            }
    }
      return ds.getConnection();
}
```

The "real" join solution

- The perfect solution is to handle joins is in only one SQL call
  - PB solves this with a feature called "preload"
- "Preloading" adds each table into the join that initially happens
  - This will be added in WebSphere/VAJ 4.0
  - Could be "faked" in BMP today
  - Would need to store a "cache" of the data read back shared by all Entities

- One more thing that I should mention is another performance trick for BMP beans. Often you have a BMP manage an object "graph", such as in an Order/LineItem solution.
- It is often more efficient to load only the "Order" information in the ejbLoad() method, and to defer loading the "LineItems" until they are actually asked for (lazy initialization)
- In this solution you would load the lineItems inside the getLineItems() method so long as they have not been previously loaded.

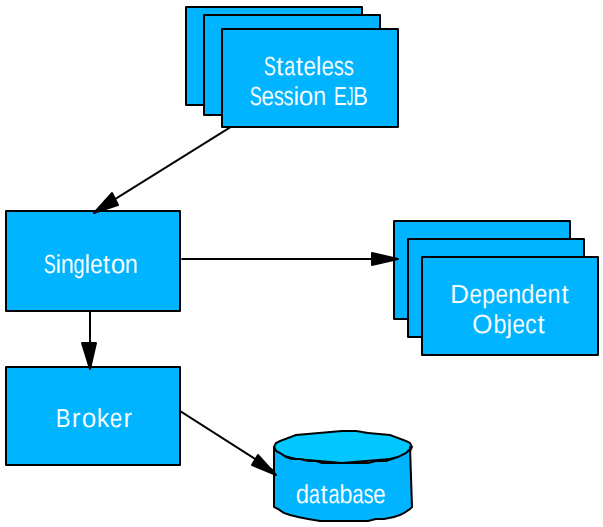
Read-only objects

- In many domains there is a set of objects whose state is frequently read but VERY rarely updated
  - Geographical locations (states, counties)
  - Codes and lookup tables
- The key is that these objects are *not* transactional
  - They do not have to be EJBs
  - Instead, make them Dependent Objects managed by a Session bean

Storing Dependent Objects

- Stateless Session beans make good "gateways" to data
  - The data itself can be cached on a per-server basis
- Use the "Singleton" pattern from *Design Patterns*
  - Store the Dependent Objects in a Singleton

Architecture of a Session Solution



Write-only objects

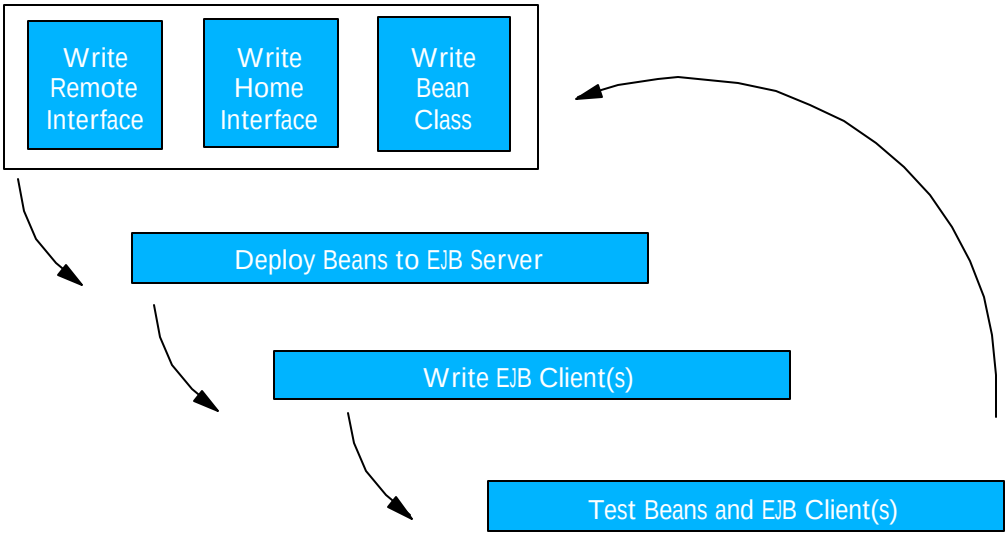
- Sometimes you often have a set of write-only objects
  - A system log
- In this case, you will not want to read the log entries
  - Inefficient to create and then not use an EJB
- A Session EJB + Dependent object solution will work well

- Be careful about logs. There are many patterns to building efficient logs. You may want to be able to route to multiple datasources for instance. Another useful option is to defer writing the logs to another process -- use an asynchronous mechanism like MQ to communicate with the write process.
- If interested, there was a good pattern language on writing logs that was submitted to PLoP 2000
- Also be sure to check out the JLog software on alphaworks (www.alphaworks.ibm.com)

Megascrolling

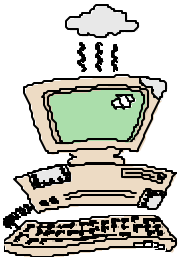
- Often you need to scroll through a large (>50 element) list
- Using an EJB to retrieve only a few fields is inefficient
- Can often code a Session bean + dependent object solution to be much faster
- Use the key value held in the dependent object to reference an Entity EJB.

EJB Development Process



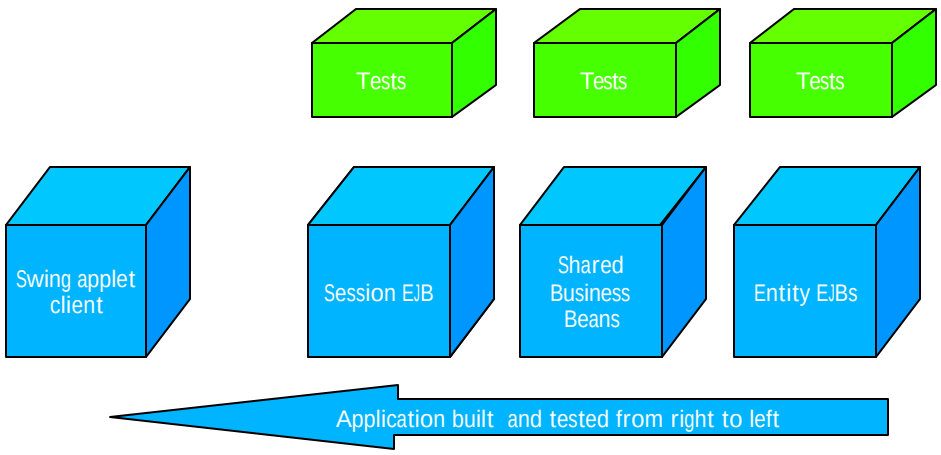
Testing

- Look into using the JUnit framework from Erich Gamma & Kent Beck
  - available at http://www.xprogramming.com
- Test all aspects of your system
  - JavaBeans
  - Back-end logic
  - Servlets
  - JSP's
  - EJB's



Layered Testing Example

Customer testing example using EJB's



- Building our application in this way had many advantages:
- (1) We were able to experiment with different ways of using EJB's without fear of bugs occurring at higher layers.
- (2) The GUI worked very nearly "the first time" because of the work we had put into testing the lower layers.

Summary

- You've seen
  - EJB Basics
  - When to use EJB's
  - Examples of using Session & Entity beans
  - Architecture and Development Best Practices

Bibliography

- Gamma, "Design Patterns", Addison-Wesley Longman, 1995
- Monson-Haefel, "Enterprise JavaBeans", O'Reilly & Sons, 1999
- See also http://members.aol.com/kgb1001001 for more links and references

WDD and VADD Articles

- Kyle Brown, "What's it going to take to get you to move to EJB's", DeveloperWorks, (http://www.ibm.com/developer)
- Vesselin Ivanov, "EJB's and Transaction Management in WebSphere Advanced Edition", VisualAge Developer's Domain, (http://www.software.ibm.com/vadd)

White Papers

- Nataraj Nagaratharam, "WebSphere 3.02 Standard/Advanced Security Overview"
- Harvey Gunther "WebSphere Application Server Development Best Practices for Performance and Scalability"
- Both available at http://www-4.ibm.com/software/webservers/appserv/whitepapers.html

Extra Topics

Transaction Demarcation

- You can specify (demarcate) transactions in three ways
  - Container Managed (implictly decide when to start/commit/rollback) transactions in the deployment descriptor
  - Bean Managed -- you explictly start/commit/rollback in EJB code
  - Client Managed -- The EJB client explicitly start/commit/rollback transactions

Transaction Attributes

	Clients TXN	TXN Associated with EJB Method
TX_NOT_SUPPORTED	-	-
	T1	-
TX_REQUIRED	-	T2
	T1	T1
TX_SUPPORTS	-	-
	T1	T1
TX_REQUIRES_NEW	-	T2
	T1	T2
TX_MANDATORY	-	ERROR
	T1	T1

- TX_REQUIRED is default in VAJ.

Advice on Transaction Demarcation

- Assume a base of TX_REQUIRED
  - All Session beans acting as facades
  - Entity beans (although TX_MANDATORY might be good here)
- In complex cases the Session bean may need TX_BEAN_MANAGED
  - Allows the Session to do exception handling and recovery logic.

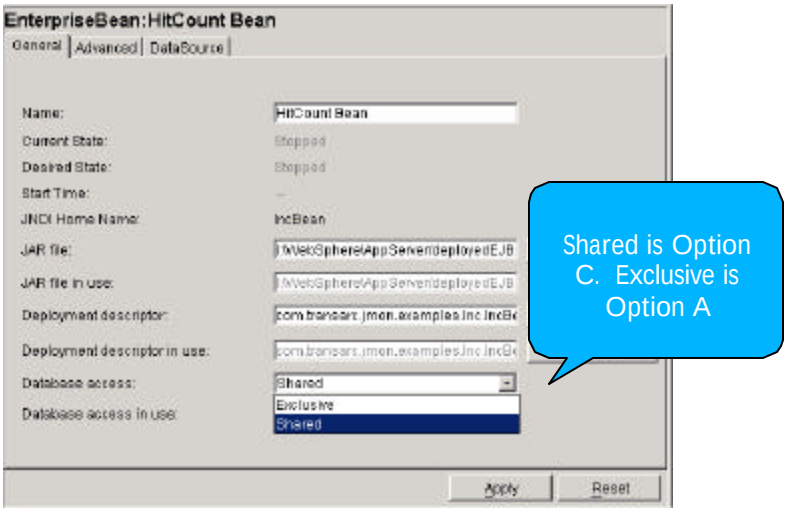
Transactions and BMP EJBs

- A BMP bean will have its ejbLoad/ejbStore methods called at the appropropriate times
- A BMP bean should never manage the transaction calls themselves
  - Should use WebSphere's connection pooling classes
  - These will automatically commit or rollback in response to EJB transactions

EJB Caching

- A way to speed up your entities is to cache them
  - By default Websphere uses "Option C"
    - This means a bean will be reloaded for every transaction
  - Can change this to "Option A"
    - This means a bean will keep its state across transactions
    - Fewer SQL SELECT statements and better performance

Caching Options in WebSphere



- You get to this page by selecting an EJB in the tree of the WebSphere Administrator's Console.

Advice on EJB Caching

- Only use Option A caching if you are CERTAIN that one EJB Server has exclusive write access to the bean
- Do not use if other processes (including other clustered EJB servers) can also update the data

Isolation Levels

- The Websphere EJB server does not provide transaction isolation and concurrency management on its own.
- It provides directives (called *Isolation levels*), which are "hints" to the database as to what policies should be used.
- The database is responsible for providing these services to your EJB's

Isolation Levels

Level	Uncommitted Data?	Nonrepeateable Reads?	Phantom Records?
TRANSACTION_READ_UNCOMMITTED	YES	YES	YES
TRANSACTION_READ_COMMITTED	NO	YES	YES
TRANSACTION_REPEATABLE_READ	NO	NO	YES
TRANSACTION_SERIALIZABLE	NO	NO	NO

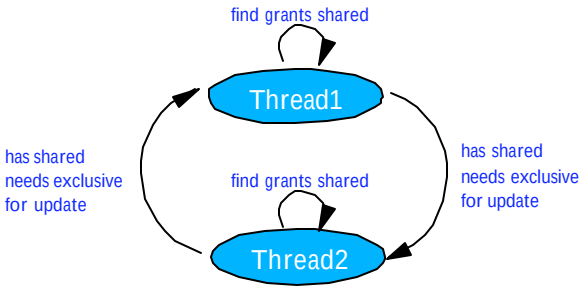
- Uncommited Data means that a transaction can read data written by another transaction that has not yet been committed.
- Nonrepeatable reads means that within a transaction .the first read of a data field can get one result, while a second can get a different result due to the data being updated by another transaction or program
- phantom records are records that can be inserted while the transaction is in progress of which this transaction may be unaware.
- The default Isolation level in WebSphere is REPEATABLE_READ

Isolation levels and deadlocks

- Choosing a high isolation level (like SERIALIZABLE) can lead to deadlocks
- Some of the options listed later can help this
- Refer to [Ivanov] for details

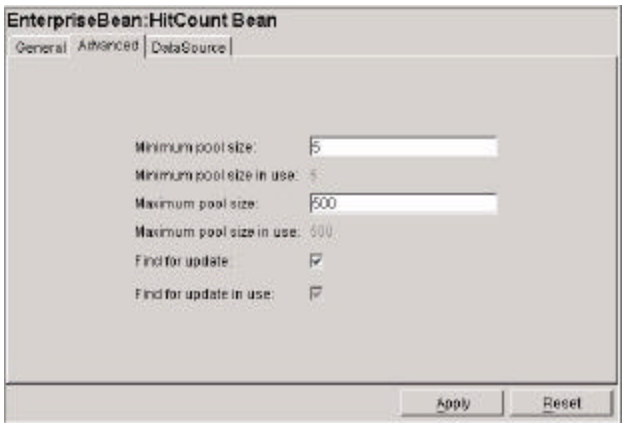
Find For Update

- Specifies whether the container gets an exclusive lock on the EJB when the "find by primary key" method is invoked
  - Useful for avoiding deadlock



- The setting will take effect the next time the application sever hosting the enterprise bean is started.
- Deadlock can occur when two transactions execute find methods, and then update methods, on the same enterprise bean.
- The find method grants a shared lock on the enterprise bean, but the update method attempts to get an exclusive lock on the enterprise bean, resulting in deadlock.

Find For Update Setting



Read-only Methods

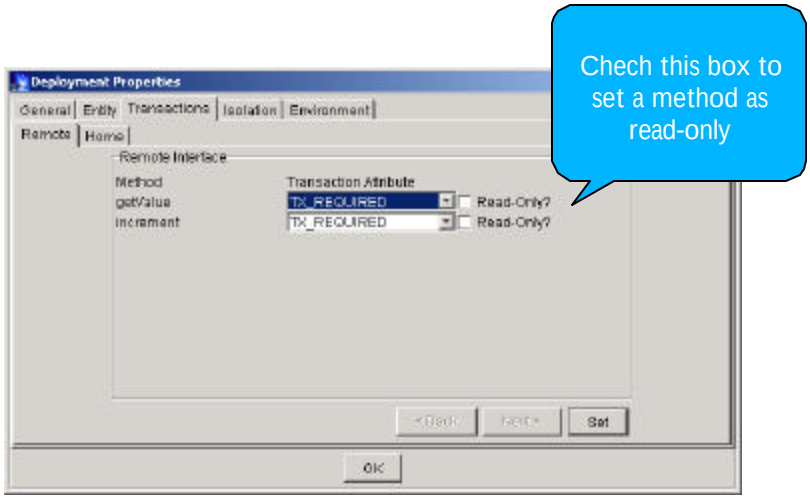
- To improve the performance of your applications consider which methods do not modify data in an Entity EJB
- The specification assumes that all beans used during a transaction are "dirty" and must have their state written back at the end of a transaction.
- WebSphere goes beyond the EJB Spec to address this issue.

- WebSphere defines a read-only method flag in the deployment descriptor of entity Beans. This allows the EJB developer to tell the container which methods are read-only, i.e., which don't change the state of the bean.

Read-only Methods *(continued)*

- WebSphere looks for the setting of this flag whenever a method is invoked.
- If only "read-only" methods are sent to a bean in a transaction, WebSphere will not assume that the bean is "dirty"
- Thus it will not execute a SQL UPDATE statement for that bean when the transaction is committed.

Setting Read-only methods



- You reach this dialog box by pressing the "Deployment Descriptor" button on the EJB page in the WebSphere Administrator's console