

---

# Software-Entwicklung mit Enterprise JavaBeans (EJB) 2.0

Prof.Dr. Jürgen Dunkel  
Fachhochschule Hannover  
Fachbereich Informatik  
juergen.dunkel@inform.fh-hannover.de

## 0. Überblick

---

1. Einleitung: SW-Architekturen
2. J2EE-Überblick
3. Grundkonzepte von EJB
4. Beispiel
5. Persistenz
6. Kritische Bewertung – EJB-Patterns

## 0. Überblick

### J2EE: Java 2 Enterprise Edition

- ⌘ Framework zur Entwicklung verteilter Anwendungen mit Java
- ⌘ J2EE-Komponenten:
  - ⌘ **client components**: Applications oder Applets
  - ⌘ **web components**: Servlets oder JSP's
  - ⌘ **business components**: Enterprise JavaBeans

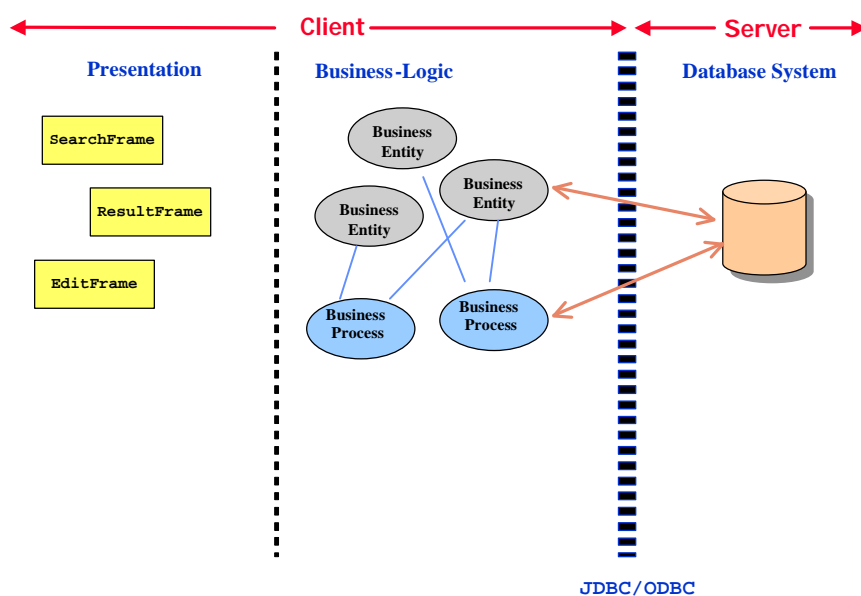
### Enterprise JavaBeans (EJB)

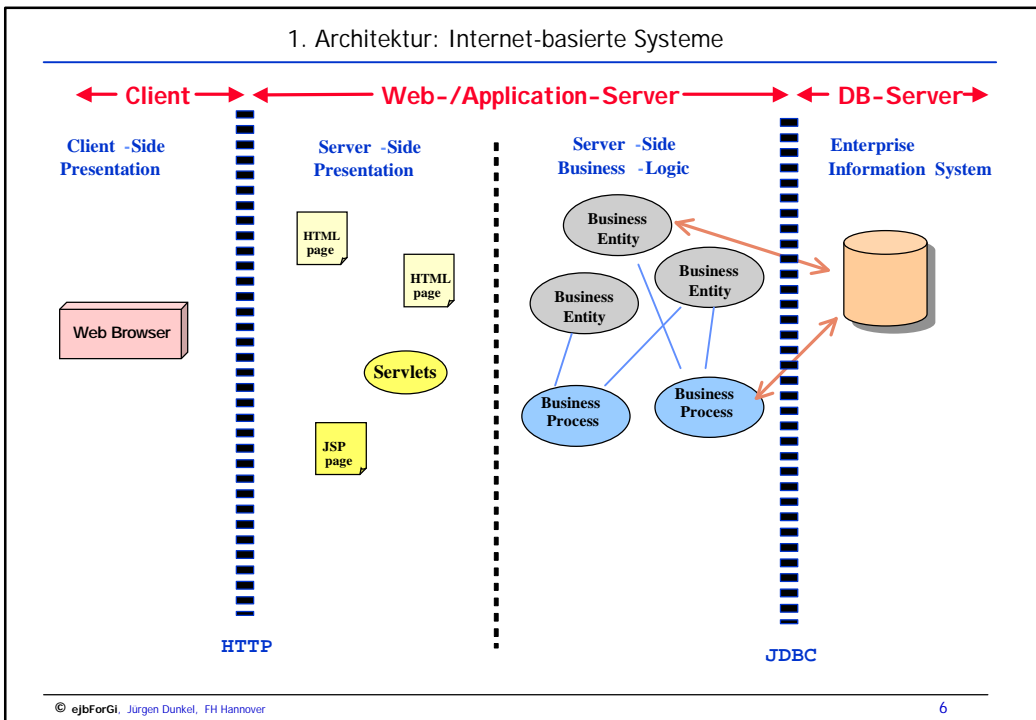
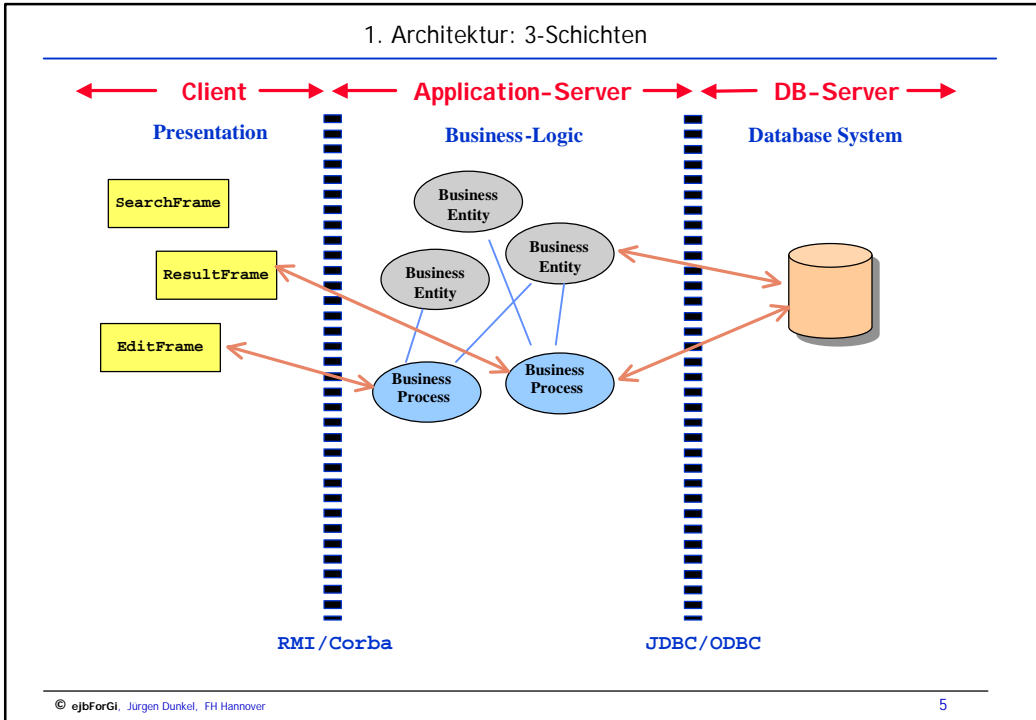
- ⌘ **Persistenz** von Java-Objekten
- ⌘ **Verteilung** von Java-Objekten

### Application-Server

- ⌘ Produkt, das J2EE-Architektur realisiert

## 1. Architektur: Client-Server



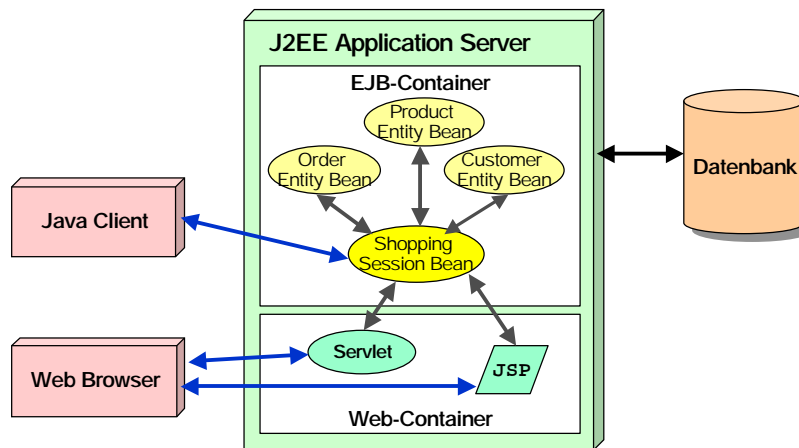


## 2. J2EE - Überblick

### Java 2 Enterprise Edition J2EE

Java-Framework zur Entwicklung von Multi-Tier-Architekturen, bietet Unterstützung von

- ☞ business components (EJB)
- ☞ web components (Servlets und JSP)



## 2. J2EE - Überblick

### J2EE Server

☞ **Application-Server** mit folgenden Diensten:

- ☞ **HTTP - Webserver**
  - ☞ Zugriff auf Servlets, Java Server Pages (JSP), HTML-Seiten
- ☞ **EJB - Container**
  - ☞ verwaltet Enterprise Java Beans
- ☞ **JNDI (Java Naming und Directory Interface)**
  - ☞ Auffinden von Diensten im Application Server über JNDI-API
- ☞ **Security: Authentication and Authorisation**
  - ☞ kontrollierter Zugriff auf Web- und Business-Komponenten

#### Produkte (Auswahl):

- BEA WebLogic, IBM WebSphere, Oracle Application Server, Sun iPlanet, JBoss, Inprise Application Server, In-Q-My.....

### EJB-Container

#### Life Cycle Management

- ⌘ EJB-Objekte durchlaufen mehrere Zustände
- ⌘ EJB-Container übernimmt die Verwaltung von Bean-Objekten (Erzeugung, Synchronisation mit DB, Pooling, Zuordnung zu Clients, ...)

#### Transaction Management

- EJB-Container übernimmt die Transaktionssteuerung

#### Database Connection Pooling

- ⌘ effiziente Verwaltung von DB-Connections

#### Security

- ⌘ nur autorisierte Clients dürfen eine EJB-Methode aufrufen

#### Remote Client Connectivity

- ⌘ EJB-Container übernimmt die Low-Level-Kommunikation zu Clients
- ⌘ genutztes Protokoll CORBA-IIOP

### Enterprise Java Beans

- ⌘ sind Java-Klassen, die EJB-Konventionen genügen:
  - ⌘ implementieren bspw. Methoden bestimmter Interfaces
- ⌘ können nur innerhalb Application Server benutzt werden
  - ⌘ müssen beim Application Server registriert werden (**Deployment**)
  - ⌘ Application-Server steuert komplett den Zugriff und die Verwaltung

#### Trennung von fachlicher Funktionalität und technischen Basisdiensten

- ⌘ Entwickler verantwortlich für: **Geschäftslogik**
- ⌘ Application-Server verantwortlich für
  - ⌘ Lifecycle-Mgt, Transactions, Security, .....
  - ⌘ Eigenschaften einer EJB werden in XML-Dateien spezifiziert
  - ⌘ im EJB-Sourcecode muß man sich nicht darum kümmern !

## Enterprise Java Beans

☞ in EJB 2.0 (Draft 2 im April 2001) gibt es drei Arten von EJB's:

- ☞ Entity Beans
- ☞ Session Beans
- ☞ Message Driven Beans

## Entity Beans (1)

- **sind daten-tragende Objekte (Business Entities)**
  - ☞ haben persistente Attribute (in Datenbank gespeichert)
    - Zugriff auf Attribute über get-/set-Methoden (EJB 2.0)
  - ☞ besitzen „Primary-Key-Attribut“
  - ☞ Bsp. eCommerce-Applikation: Buch, Kunde, Auftrag, Rechnung
- **kapseln den Datenzugriff (auf DB-Tabellen)**
  - ☞ redundanzfreier SQL-Zugriff
  - ☞ fachliche Methode für konsistenten Datenzugriff
- **werden von mehreren Clients parallel genutzt**
  - ☞ ermöglichen Transaktionen

## Entity Beans (2)

Frage: wer erzeugt die SQL-Befehle für den Datenbankzugriff ?

### A. Container-Managed-Persistence (CMP)

- der EJB-Container generiert vollständig den SQL-Code
- in XML-Datei werden EJB-Attribute auf DB-Tabellenspalten abgebildet
- sehr starke Erweiterung mit EJB 2.0: u.a. Verwaltung von Beziehungen

### B. Bean-Managed-Persistence (BMP)

- Entwickler schreibt Zugriffscode (bspw. „normaler“ SQL/JDBC-Code)
- datenbank-spezifische Zugriffe (stored procedures, Legacy Systems, hints,...)
- in EJB 1.1 ist BMP bei many-many-Beziehungen erforderlich

## Session Beans

- für **Geschäftsprozesse (Business Processes)**
  - ⚡ Methoden, die komplexere Aufgabe ausführen
    - nutzen ggf. andere Entity- und Session-Beans
    - bilden eine Transaktions-Klammer
  - ⚡ besitzen keine persistenten Attribute
  - ⚡ kapseln ggf. die Entity-Beans vor den Clients
    - bieten den Clients eine einfachere Sicht
  - ⚡ Bsp. eCommerce-Applikation: BestellEingang
- Sonderform: **stateful Session Beans**
  - ⚡ haben Attribute, die einem Client zugeordnet und an dessen Lebensdauer gekoppelt sind
  - ⚡ für Session Tracking (bspw. Einkaufswagen)

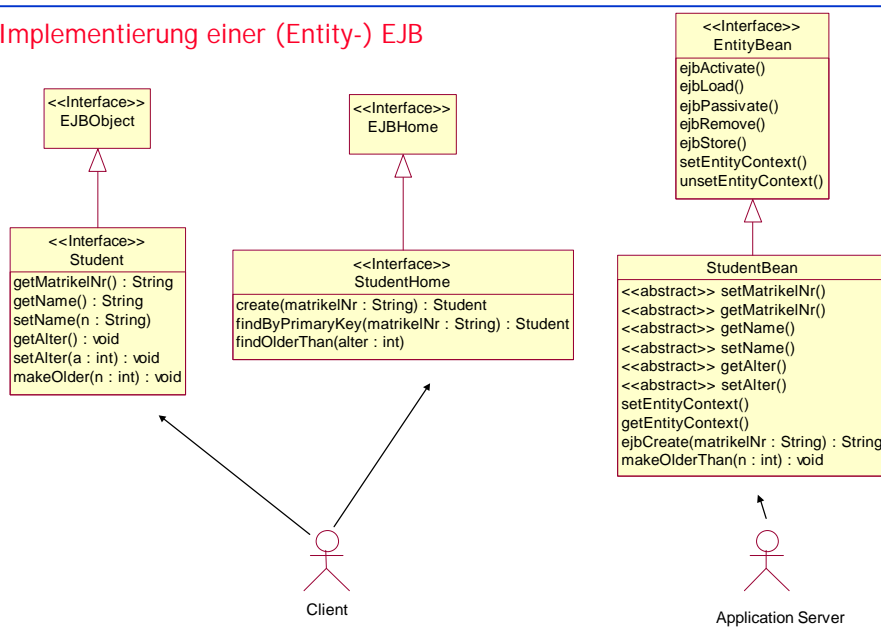
### 3. EJB - Grundkonzepte

#### Message Driven Beans

- für **Verarbeitung asynchroner Nachrichten**
  - ⚡ Message-Listener: wartet auf den Eingang von Nachrichten
    - es gibt verschiedene Arten von Nachrichten (ObjectMessage, TextMessage,...)
    - Container ruft die Methode onMessage() auf
  - ⚡ gibt's erst mit EJB 2.0
  - ⚡ werden eher selten eingesetzt

### 4. Beispiel

#### Implementierung einer (Entity-) EJB



## 4. Beispiel

### Remote Interface

```
import java.rmi.RemoteException;
import javax.ejb.EJBObject;

public interface Student extends EJBObject {
    public String getMatrikelNr() throws RemoteException;

    public String getName() throws RemoteException;
    public void setName(String str) throws RemoteException;
    .....
    public int makeOlder(int anzahlJahre) throws RemoteException;
}
```

- ⌘ **spezifiziert alle fachlichen Methoden eines Student-Beans**
  - ⌘ Client verwendet nur die Methoden des Remote-Interface
- ⌘ Methoden werden **remote** aufgerufen
  - ⌘ werfen deshalb **RemoteException (oder auch eigene fachliche)**
  - ⌘ genutztes Protokoll ist allerdings nicht RMI, sondern Corba IIOP

## 4. Beispiel

### Home Interface

```
import javax.ejb.*;
import java.rmi.RemoteException;
import java.util.Collection;

public interface StudentHome extends EJBHome {

    public Student create(String matrikelNr)
        throws CreateException, RemoteException;

    public Student findByPrimaryKey(String matrikelNr)
        throws FinderException, RemoteException;

    public Collection findOlderThan(int alter)
        throws FinderException, RemoteException;
}
```

- ⌘ **bietet Methoden zur Erzeugung u. Suche von Entity-Bean-Objekten**
  - ⌘ **create**(int oid) erzeugt ein neues Studentenobjekt
  - ⌘ **findByPrimaryKey**(String oid) holt ein Studentenobjekt
  - ⌘ ggf. weitere Such-Methoden **Collection findOlderThan(int alter)**

#### 4. Beispiel

### Entity Bean Klasse

```
abstract public class StudentBean implements EntityBean {

    private EntityContext ctx;
    public StudentBean() {}

    public void setEntityContext(EntityContext ctx) { this.ctx = ctx; }
    public void unsetEntityContext() { this.ctx = null; }

    abstract public void setMatrikelNr(String matrikelNr);
    abstract public String getMatrikelNr();
    abstract public void setName(String str);
    abstract public String getName(); ...

    public String ejbCreate(String matrikelNr) throws CreateException {
        this.setMatrikelNr(matrikelNr);
        return null;
    }

    public int makeOlder(int anzahlJahre) {
        this.setAlter(this.getAlter + anzahlJahre);
        return this.getAlter();
    }

    // leere HookMethoden
    public void ejbActivate() { }
    public void ejbPassivate() { }
    public void ejbLoad() { }
    public void ejbStore() { }
    public void ejbRemove() throws RemoveException { }
    public void ejbPostCreate(String matrikelNr) { }
}
```

#### 4. Beispiel

### Deployment von EJB's drei XML-Descriptors

#### 1. ejb-jar.xml : beschreibt Bean auf logischer Ebene

- ✗ Namen der Java-Klassen
- ✗ Namen der Attribute, welches ist der Primary Key
- ✗ spezifizieren Query-Methoden

```
<ejb-jar>
  <enterprise-beans>
    <entity>
      <ejb-name>ejbStudent</ejb-name>
      <home>StudentHome</home>
      <remote>Student</remote>
      <ejb-class>StudentBean</ejb-class>

      <persistence-type>Container</persistence-type>
      <prim-key-class>java.lang.String</prim-key-class>
      <reentrant>False</reentrant>
      <cmp-version>2.x</cmp-version>

      <abstract-schema-name>StudentEJB</abstract-schema-name>
      <cmp-field> <field-name>matrikelNr</field-name> </cmp-field>
      <cmp-field> <field-name>name</field-name> </cmp-field>
      <primkey-field>matrikelNr</primkey-field>
    </entity>
  </enterprise-beans>
</ejb-jar>
```

#### 4. Beispiel

```
<query>
  <query-method>
    <method-name>findOlderThan</method-name>
    <method-params>
      <method-param>int</method-param>
    </method-params>
  </query-method>

  <ejb-ql>
    <![CDATA[WHERE alter > ?1]]>
  </ejb-ql>
</query>
</entity>
</enterprise-beans>
<assembly-descriptor>
  <container-transaction>
    <method>
      <ejb-name>ejbStudent</ejb-name>
      <method-intf>Remote</method-intf>
      <method-name>* </method-name>
    </method>
    <trans-attribute>Required</trans-attribute>
  </container-transaction>
</assembly-descriptor>
</ejb-jar>
```

#### 4. Beispiel

##### XML-Descriptor (2)

##### 2. weblogic-ejb-jar.xml : beschreibt Informationen für den Application Server

- ⌘ Versionsnummern,...
- ⌘ **jndi-Name**: Name unter dem das Home-Interface gefunden wird

```
<weblogic-ejb-jar>
  <weblogic-enterprise-bean>
    <ejb-name>    ejbStudent  </ejb-name>
    ....
    <persistence-type>
      <type-identifier>WebLogic_CMP_RDBMS</type-identifier> ...
      <type-storage>META-INF/weblogic-cmp-rdbms-jar.xml</type-storage>
    </persistence-type>
  </persistence-type>
</entity-descriptor>

  <jndi-name>myStudentHome</jndi-name>

</weblogic-enterprise-bean>

</weblogic-ejb-jar>
```

## 4. Beispiel

### XML-Descriptor (3)

#### 3. weblogic-cmp-rdbms-jar.xml : O/R-Mapping für die Datenbank

- ⚡ **Datasource:** Information für den DB-Connect im Application Server
- ⚡ **jndi-Name:** Name unter dem das Home-Interface gefunden wird

```
<weblogic-rdbms-jar>
<weblogic-rdbms-bean>
  <ejb-name>ejbStudent</ejb-name>
  <data-source-name>dataSource-oraclePool</data-source-name>
  <table-name>ejbStudent</table-name>
    <field-map>
      <cmp-field>matrikelNr</cmp-field>
      <dbms-column>matrikelNr</dbms-column>
    </field-map>
    <field-map>
      <cmp-field>name</cmp-field>
      <dbms-column>name</dbms-column>
    </field-map>
    <field-map>
      <cmp-field>alter</cmp-field>
      <dbms-column>jahre</dbms-column>
    </field-map>
  </weblogic-rdbms-bean>
</weblogic-rdbms-jar>>
```

## 4. Beispiel

### EJB Client (1)

1. stellt **Verbindung zum Application Server** her
2. erhält **über jndi-Namen** (java naming and directory service ) **Zugriff auf Home-Interface**
3. nutzt **Home-Interface, um Objekte zu finden oder neu zu erzeugen**
4. ruft für die Objekte die **Methoden des Remote-Interface auf**

```
try {
  Properties h = new Properties();
  h.put(Context.INITIAL_CONTEXT_FACTORY, "weblogic.jndi.WLInitialContextFactory");
  h.put(Context.PROVIDER_URL, url);
  Context ctx = new InitialContext(h);

  // jndi-Name aus weblogic-ejb-jar.xml verwenden !!!
  Object obj = ctx.lookup("myStudentHome");
  home = (StudentHome) PortableRemoteObject.narrow(obj, StudentHome.class);
} catch (NamingException e){System.out.println("unable to find EJBHome");}
```

- ⚡ **"jndiPerson"** ist der Name mit dem man im Application-Server das PersonBean registriert hat

## EJB Client (2)

```
try {  
    // erzeuge neues EntityBean-Objekt  
    s = home.create("4");  
    s.setAlter(32); s.setName("Sam");  
  
    // hole 1 Objekt aus Application Server  
    s = home.findByPrimaryKey("4711");  
    s.makeOlder(13);  
  
    // hole viele Objekte aus Applikation Server  
    Collection col = home.findOlderThan(20);  
    Iterator it = col.iterator();  
    while (it.hasNext()) {  
        s = (Student) it.next();  
        System.out.println(s.getString());  
    }  
} catch (Exception e) {e.printStackTrace(); }
```

## Session Beans

⚡ Implementierung analog zu Entity Beans

⚡ Unterschiede der Implementierung:

- ⚡ kein Primary Key
- ⚡ meist nur eine create-Methode ohne Parameter
- ⚡ keine find-Methoden
- ⚡ weniger Hook-Methoden (ejbStore, ejbLoad,...)

## 5. Persistenz

### 1. Datenbank-Connections

- ☞ vom EJB-Container verwaltet (Pooling)

### 2. Synchronisation: Entity-Bean ☞ ☞ Datenbank

#### ☞ Hook-Methoden

- ☞ führen den **SQL-Zugriff durch**
- ☞ müssen implementiert werden (sind definiert im Interface EntityBean)

#### ☞ Aufrufe der Hook-Methoden durch den EJB-Container

- ☞ sind an den Lebenszyklus der Entity-Beans gekoppelt
- ☞ immer, **wenn sich ein Entity-Bean-Objekt ändert**
  - ☞ vor Aufruf einer Methode : Objekt-Attribute aus Datenbank holen
  - ☞ nach Beendigung der Methode: update der Datenbank

Vorteil: immer konsistent mit Datenbank

Nachteil: großer Aufwand, schlechte Performanz

## 5. Persistenz: Hook-Methoden

Methoden	SQL-Befehl	Aufrufzeitpunkt
<code>ejbCreate()</code>	<b>insert</b>	Client ruft <code>create()</code> auf
<code>ejbStore()</code>	<b>update</b>	nach Ausführung fachlicher Methode
<code>ejbLoad()</code>	<b>select</b>	vor Ausführung fachlicher Methode
<code>ejbRemove()</code>	<b>delete</b>	Client ruft <code>remove()</code> auf
<code>ejbFindByXy(...)</code>	<b>spezielles select</b>	Client ruft <code>findByXy()</code> auf

#### bei Bean-Managed-Persistence (BMP):

- ☞ in Hook-Methoden SQL-Zugriffcode implementieren

#### bei Container-Managed-Persistence (CMP):

- ☞ Hook-Methoden mit leerem Methodenrumpf

### 3. Transaktionen

#### ☞ **Transaktions-Attribute**

☞ werden für fachliche Methoden im Deployment-Descriptor festgelegt

#### **REQUIRED (Standard-Einstellung)**

☞ wird die Methode innerhalb einer Transaktion t1 aufgerufen, so gehört sie dazu

☞ sonst: eine neue Transaktion wird gestartet

```
geheZurKasse(Warenkorb w, Kunde k) {  
    erstelleRechnung(w, k);  
    veranlasseLieferung(w, k);  
}
```

☞ haben alle drei Methoden das Attribut REQUIRED, dann finden sie innerhalb einer Transaktion statt

#### **RequiresNew**

☞ startet auf **jeden Fall eine neue eigene Transaktion T2** (für nested Transactions)

.....

### 4. Verwaltung von Beziehungen

#### ☞ **m:n-Beziehungen sind mit CMP möglich (ab EJB 2.0 !)**

☞ SQL für Zugriffe wird generiert

☞ lazy Initialization (loading on demand)

```
interface Customer extends EJBObject {  
    void addAccount (Account a);  
    Collection getAccounts();  
}
```

#### **in CustomerBean-Klasse:**

☞ die beiden Methoden werden nur als abstract deklariert (wie im Interface)

#### **in Deployment-Deskriptoren:**

☞ Account wird als cmr(container-managed-relationship)-Feld festgelegt

☞ Kardialität und Ziel-Entity-Bean wird angegeben

☞ beim O/R-Mapping wird der Beziehung die entsprechende Beziehungstabelle (inkl. Spalten) zugeordnet

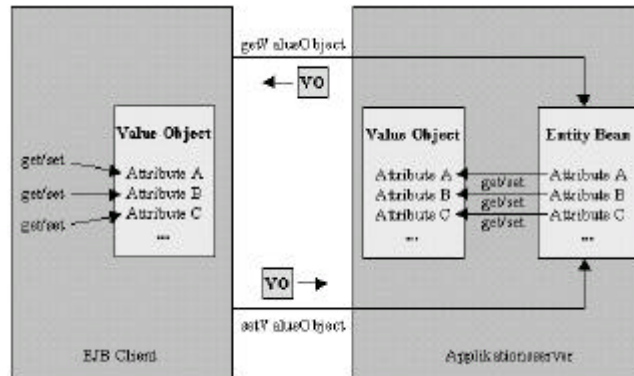
## 6. Kritische Bewertung

### Probleme von J2EE (1)

#### ⌘ Performance

- ⌘ Zugriff übers Netz per Corba-IIOP ist teuer
  - ⌘ Overhead durch Marshalling, Transaktionen, Security
- ⌘ sehr häufiger Zugriff bei jedem Methodenaufruf

**LÖSUNG: Object-Value-Pattern** reduziert Zahl der Remoteaufrufe



## 6. Kritische Bewertung

### Probleme von J2EE (2)

#### ⌘ Modelle werden komplexer

- ⌘ jede fachliche Klasse wird auf drei EJB-Klassen/Interfaces abgebildet
- ⌘ Patterns führen teilw. zu mehr Klassen und Methoden
- ⌘ kein klares Vorgehen (Best Practices fehlen)

#### ⌘ der Java-Sourcecode wird von technischem Code befreit, aber

- ⌘ Konsistenz von Interfaces und Bean-Klassen
- ⌘ Deployment-Deskriptoren werden sehr komplex
- ⌘ debugging schwierig

#### ⌘ Compilierung-Debugging- Deployment ist komplex und langsam

- ⌘ viele Systeme beteiligt: keine Durchgängigkeit

#### ⌘ wenig Unterstützung für das Presentation Layer

## Probleme von J2EE (3)

### LÖSUNGSANSÄTZE:

#### Modellierung

- ⚡ Erweiterung von UML
- ⚡ Design-Richtlinien und (EJB-)Patterns
- ⚡ Vorgehen definieren

#### Generierung der Interfaces und Deployment-Deskriptoren

- ⚡ „Anreicherung“ bereits bei der Modellierung (OMG/MDA - Model driven Architecture) (bspw. ArcStyler,...)
- ⚡ angepasste IDE's
- ⚡ Opensource-Produkt EJGen

#### Automatisierung des Deployment-Prozesses

- ⚡ Werkzeugunterstützung, bspw. Apache/ant oder IDE's

## Resümee

### Stärken:

- ⚡ komplette **Infrastruktur zur Entwicklung verteilter Systeme**
  - ⚡ de-facto-Standard, viele große Anbieter
- ⚡ bietet **komplette SW-Architektur**
- ⚡ **Entkoppelung fachlicher Funktionalität von Datenbank**
  - ⚡ Client enthalten keinen SQL-Code
  - ⚡ SQL-code ist redundanzfrei
  - ⚡ Entwickler muß sich nicht um Synchronisation mit DB kümmern
- ⚡ **befreit Entwickler von technischen Basisdiensten**
  - ⚡ Naming, Security, Transaktionen
  - ⚡ manchmal nicht ausreichend ?

### Probleme:

- ⚡ Performance
- ⚡ komplexere Modelle
- ⚡ Leitlinien fürs Vorgehen
- ⚡ komplexer Deployment-Prozess