

EJB 3 und MDSD

...geht da was?



**Präsentation SE2007-Konferenz (Workshop Model Driven Software Development Today),
Hamburg, 28.03.2007**

Oliver Ihns, Holisticon AG

Über mich...

- Mit-Gründer und Vorstand (CTO) der Holisticon AG – Management- und IT-Unternehmensberatung
- Mitglied der EJB 3.0 Expert Group (JSR220), Mitglied der WebBeans Expert Group (JSR299) im JCP bei Sun Microsystems
- Von Sun berufener „Java Champion“ (kleine Gruppe um James Gosling herum)
- Herausgeber und einer der Autoren des Fachbuches „Enterprise JavaBeans komplett – EJB 2.1“
- Herausgeber und einer der Autoren des Fachbuches „EJB 3 professionell“ (Veröffentlichung: April 2007)
- Autor diverser Fachartikel (JavaMagazin, IX, ObjektSPEKTRUM,...)
Sprecher auf div. Konferenzen (OOP, JAX, Java Forum Stuttgart, ...)
- Pers. Fokus: Software-Architekturen und Technologien für verteilte Softwaresysteme



Oliver Ihns

oliver.ihns@holisticon.de



Oldenbourg
Wissenschafts-
Verlag

dpunkt Verlag

Agenda – im Überblick

EJB 2.x und modellgetriebene Entwicklung

Rückblick EJB 2.x

"Zurück" zu POJO/POJI

Verwendung von Meta-Annotationen

Kapselung von Umgebungs- und JNDI-Zugriffen

Weitere Neuerungen / Erleichterungen

EJB 3.0

Session Bean

Message-driven Bean

Persistent Entity / Persistenz

Fazit

Agenda



EJB 2.x und modellgetriebene Entwicklung

"Zurück" zu POJO/POJI

Verwendung von Meta-Annotationen

Kapselung von Umgebungs- und JNDI-Zugriffen

Weitere Neuerungen / Erleichterungen

Session Bean

Message-driven Bean

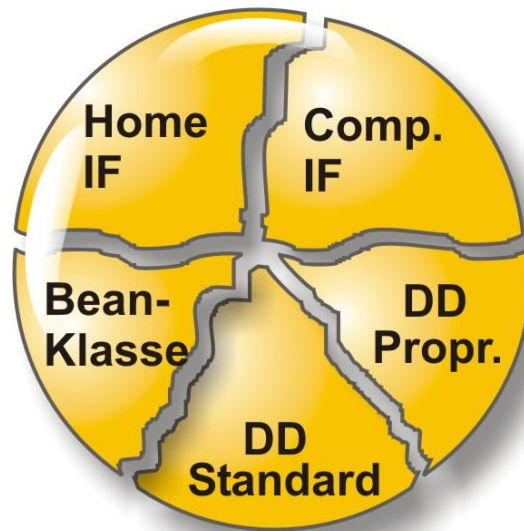
Persistent Entity / Persistenz

Fazit

EJB 2.x und modellgetriebene Entwicklung

- Die Mikroarchitektur der EJB-Komponenten: das Grundproblem
 - Mind. 5 z.T. komplexe Artefakte → eher 7 oder mehr
 - Viele Abhängigkeiten untereinander

EJB 2.x



EJB 2.x und modellgetriebene Entwicklung

EJB 2.x-Komponente

Component Interface

```
public interface ShoppingBasket extends EJBObject {  
    public void addProduct(int prodID, int quantity)  
        throws RemoteException;  
    public void removeProduct(int prodID)  
        throws RemoteException;  
}
```

Bean-Klasse

```
public class ShoppingBasketBean implements SessionBean {  
    private SessionContext ctx;  
    public void setSessionContext(SessionContext ctx) {  
        this.ctx = ctx;  
    }  
    public void ejbCreate () {}  
    public void ejbActivate () {}  
    public void ejbPassivate () {}  
    public void ejbRemove() {}  
  
    public void addProduct (int prodID, int quantity) {  
        ...  
    }  
    public void removeProduct (int prodID) {  
        ...  
    }  
}
```

Home Interface

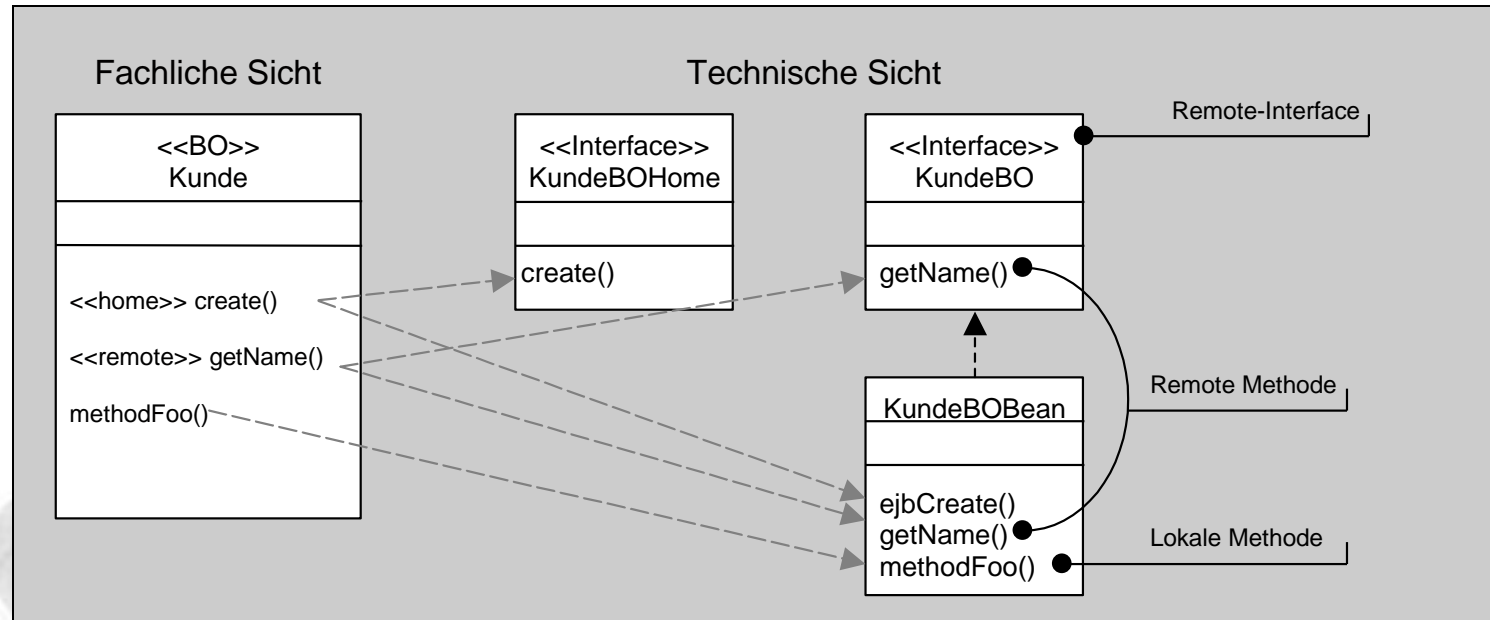
```
public interface ShoppingBasketHome extends EJBHome {  
    ShoppingBasket create() throws CreateException,  
        RemoteException;  
}
```

Deployment Deskriptor

```
...  
<session>  
    <ejb-name>ShoppingBasketEJB</ejb-name>  
    <home>com.example.ShoppingBasketHome</home>  
    <remote>com.example. ShoppingBasket</remote>  
    <ejb-class>com.example. ShoppingBasketBean</ejb-class>  
    <session-type>Stateless</session-type>  
    <transaction-type>Container</transaction-type>  
</session>  
...
```

EJB 2.x und modellgetriebene Entwicklung

- Das Problem der technischen Komplexität des Frameworks
 - Fachliche vs. technische Modellierung



EJB 2.x und modellgetriebene Entwicklung

- Das Problem der technischen Komplexität des Frameworks
 - Deployment-Deskriptoren (sehr komplex, mehrere)
 - Starke Durchsetzung des Frameworks mit technischen Elementen (RMI, IIOP, technische Exceptions etc.)
 - Vorgegebene Vererbung (technischer Interfaces) → komplexe Mikroarchitektur
 - Home Interface muss von { EJBHome | EJBLocalHome } erben
 - Component Interface muß von { EJBObject | EJBLocalObject } erben
 - Bean-Klasse musste eines der Interfaces
{ SessionBean | MessageDrivenBean | EntityBean }
implementieren
 - Bean-Klasse musste div. weitere Methoden mit teilvorgegebenen Signaturen implementieren, die in keinem Interface vorgegeben waren
 - U.a. Keine Prüfungsmöglichkeiten zur Kompilierungszeit

EJB 2.x und modellgetriebene Entwicklung

- Problem mit der nicht vollständigen Objektorientierung
 - Entity Beans konnten keine Vererbung und Polymorphie
- Ergebnis
 - Verwendung von MDSD im Kontext EJB 2.x war nicht soooo groß
 - Es gab/gibt Versuche
 - XDoclet: einfache Generierung... aber kein MDSD
 - Middlegen: einfache Generierung aus DB-Modellen... kein MDSD
 - EJBGen: einfache Generierung aus JavaDocs... kein MDSD
 - UML2EJB → Heute: AndroMDA: schon besser... ;-)

■ Fazit

- EJB 2.x → sehr komplex
- Eignete sich nicht so gut, um damit zu modellieren
- MDSD und EJB 2 ließ sich nicht so ganz verheiraten



Agenda

EJB 2.x und modellgetriebene Entwicklung

"Zurück" zu POJO/POJI

 Verwendung von Meta-Annotationen

Kapselung von Umgebungs- und JNDI-Zugriffen

Weitere Neuerungen / Erleichterungen

Session Bean

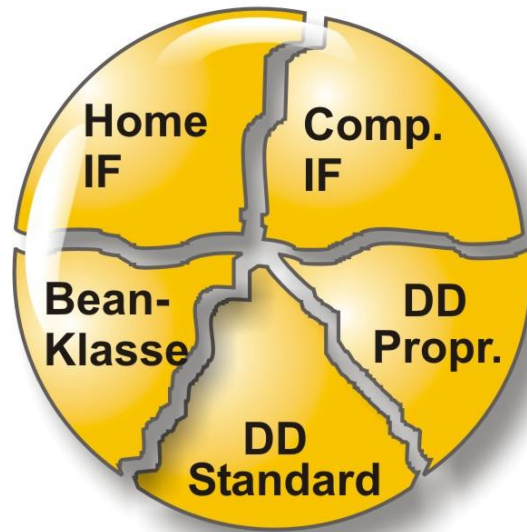
Message-driven Bean

Persistent Entity / Persistenz

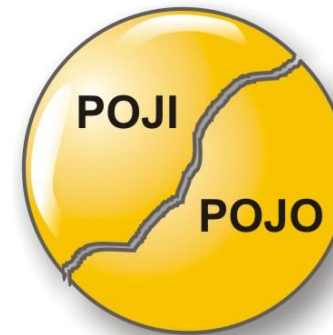
Fazit

„Zurück“ zu POJO/POJI

EJB 2.x



EJB 3.0



„Zurück“ zu POJO/POJI

Beispiel: EJB 3.0 - POJO + POJI

Business Interface

```
@Remote public interface ShoppingBasket {  
    public void addProduct(int prodID, int quantity);  
  
    public void removeProduct(int prodID);  
  
}
```

Bean-Klasse

```
@Stateful public class ShoppingBasketBean  
    implements ShoppingBasket {  
  
    public void addProduct(int productID, int quantity) {  
        ...  
    }  
  
    public void removeProduct(int productID) {  
        ...  
    }  
  
}
```

Home Interface



Deployment Descriptor



Agenda

EJB 2.x und modellgetriebene Entwicklung

"Zurück" zu POJO/POJI

Verwendung von Meta-Annotationen



Kapselung von Umgebungs- und JNDI-Zugriffen

Weitere Neuerungen / Erleichterungen

Session Bean

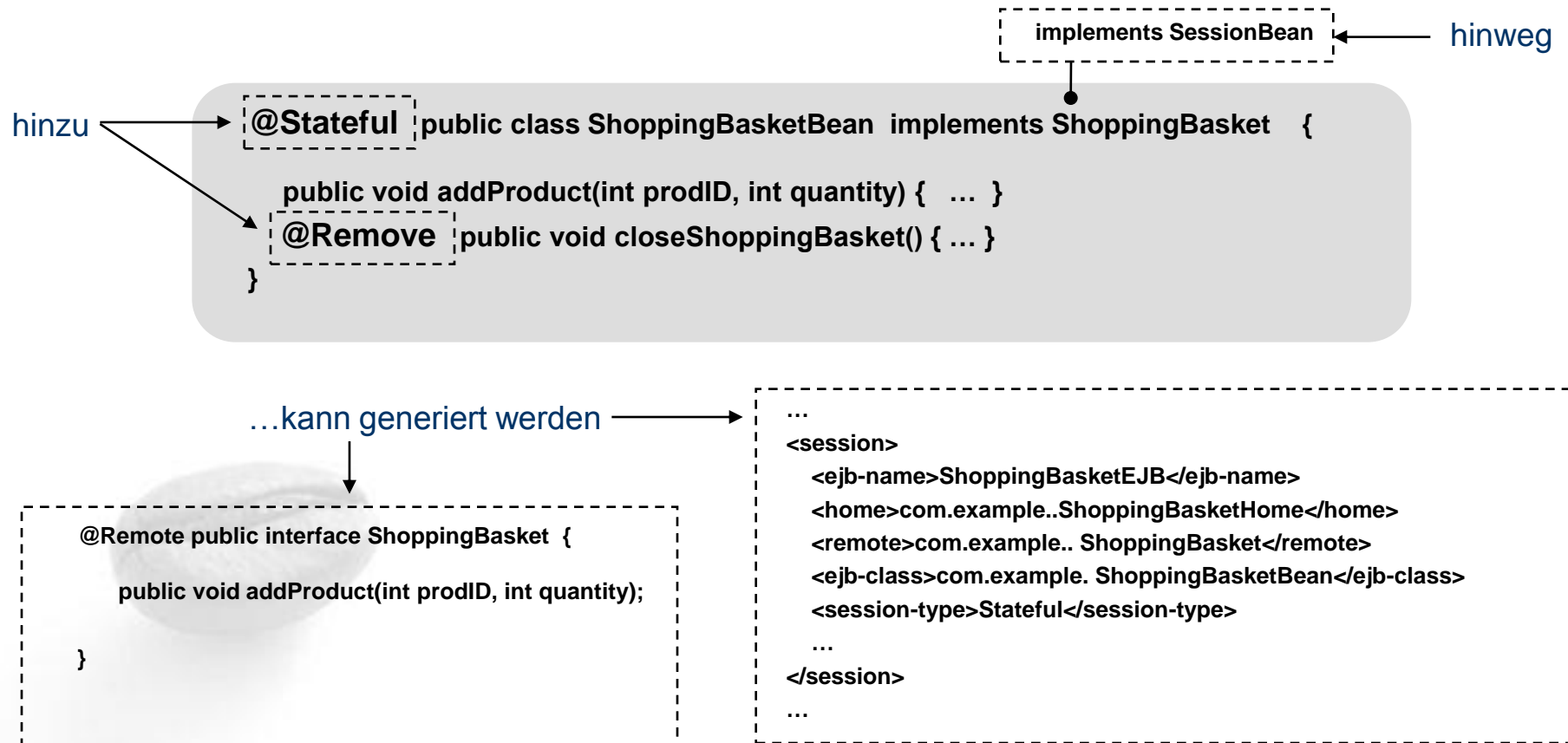
Message-driven Bean

Persistent Entity / Persistenz

Fazit

Meta-Annotationen

- Verwendung von Meta-Annotationen im Java-Quellcode
 - ▶ Siehe JSR 175 (A Metadata Facility for the Java™ Programming Language)
 - ▶ XDoclet lässt grüßen



Meta-Annotationen

- Meta-Annotationen ermöglichen der EJB-Technologie...
 - ▶ Generierung von Interfaces
 - ▶ Steuerung des Laufzeitverhaltens von EJB-Komponenten
 - ▶ Eliminierung der Deployment Deskriptoren
 - ▶ Demarkation von Injektions-Punkten

- Annotiert werden...
 - ▶ Interfaces
 - ▶ Klassen
 - ▶ Methoden
 - ▶ Attribute



Agenda

EJB 2.x und modellgetriebene Entwicklung

"Zurück" zu POJO/POJI

Verwendung von Meta-Annotationen

Kapselung von Umgebungs- und JNDI-Zugriffen

Weitere Neuerungen / Erleichterungen

Session Bean

Message-driven Bean

Persistent Entity / Persistenz

Fazit

Umgebungs- und JNDI-Zugriffe

- ...durch Einführung von
 - ▶ „Inversion of Control“
 - „Don't call us, we call u“ (Hollywood-Prinzip)
 - ▶ „Dependency Injection“
- EJB-Container initialisiert annotierte Attribute bzw. Methoden automatisch mit Referenzen auf die gewünschten Ressourcen
 - ▶ Geschieht vor der ersten Ausführung einer Geschäftsmethode
- Demarkiert durch Meta-Annotationen



Umgebungs- und JNDI-Zugriffe

- Vorteil: es muss weniger bis gar kein Code mehr geschrieben werden, um auf Ressourcen zuzugreifen

- Vorteil: ermöglicht vereinfachtes Testen von EJB-Komponenten
 - ▶ J2EE-Applikationsserver als Testumgebung nicht mehr zwingend notwendig
 - ▶ Über **setter**-Methoden werden einer Komponente zur Laufzeit entsprechende Informationen zur Verfügung gestellt (injiziert).
 - ▶ „Dependency Injection“ (DI)



Agenda

EJB 2.x und modellgetriebene Entwicklung

"Zurück" zu POJO/POJI

Verwendung von Meta-Annotationen

Kapselung von Umgebungs- und JNDI-Zugriffen

Weitere Neuerungen / Erleichterungen



Session Bean

Message-driven Bean

Persistent Entity / Persistenz

Fazit

Weitere Neuerungen in EJB 3.0

■ "Configuration by Exception"

- ▶ Spezifiziertes Default-Verhalten von EJB-Komponenten
- ▶ Ziel: Reduzierung der Entwicklungszeit
- ▶ Beispiele:

– Session Beans sind per default „lokal“ zugreifbar

- Explizite Steuerung durch Verwendung der Annotationen
`@Local` und `@Remote` für das Interface bzw. die Bean-Klasse

– Automatische Ableitung der Zugriffsart auf Attribute von Entity Beans

- Zugriff kann Field-based Access oder Property-based Access erfolgen

`Field-based == direkter Zugriff auf Attribute`

`Property-based == Zugriff über getter/setter-Methoden`

- Aus der Annotation eines Attributes bzw. einer getter-Methode erfolgt Ableitung der Zugriffsart

Weitere Neuerungen in EJB 3.0

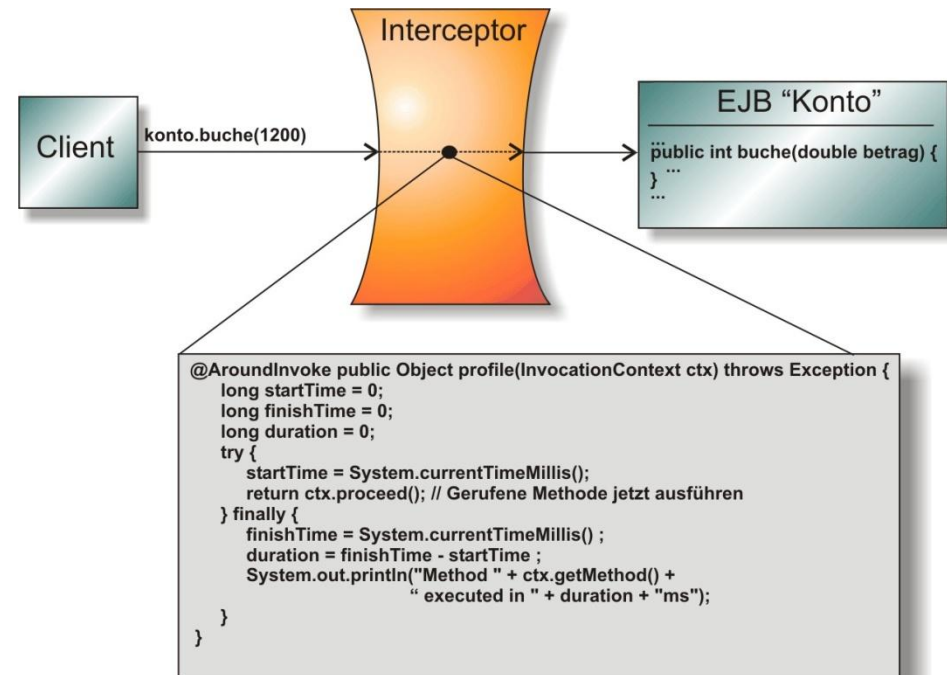
- Callback-Annotationen für Lifecycle Events
 - ▶ Vordefinierte Callbacks
 - ▶ Optional nutzbar – keine Verpflichtung der Angabe
 - ▶ Gekennzeichnet durch Annotationen
(z.B. `@PostConstruct`, `@PreDestroy`)
 - ▶ Session Beans, Message-driven Beans, Entity Beans



Weitere Neuerungen in EJB 3.0

■ Interceptor-Methoden und -Klassen

- ▶ Abfangen von Geschäftsmethodenaufrufen
- ▶ Transparent für rufenden Client und EJBs
- ▶ Ermöglicht die Ausführung von beliebigem Code
 - **Z.B. technische Prüfungen, Security, Tracking**
- ▶ Session Beans, Message-driven Beans
- ▶ **AOSD hält Einzug!**
 - **Aspektorientierte Programmierung**
 - Zumindest ein Hauch davon. Auch hier gibt es kontroverse Diskussionen wie stark aspektorientiert EJB 3 ist.



Agenda

EJB 2.x und modellgetriebene Entwicklung

"Zurück" zu POJO/POJI

Verwendung von Meta-Annotationen

Kapselung von Umgebungs- und JNDI-Zugriffen

Weitere Neuerungen / Erleichterungen

Session Bean

Message-driven Bean

Persistent Entity / Persistenz

Fazit

Session Bean

- Home Interface entfällt
 - ▶ Zugriff aus Sicht der Clients vereinfacht

```
public static void main(...) throws Exception {  
    InitialContext ctx = new InitialContext();  
    Object o = ctx.lookup(„ShoppingBasket“);  
    ShoppingBasketHome home =  
        (ShoppingBasketHome) PortableRemoteObject.narrow(o, ShoppingBasketHome.class);  
  
    ShoppingBasket sb = home.create();  
  
    ...// do something  
}
```

2.1

```
public static void main(...) throws Exception {  
    InitialContext ctx = new InitialContext();  
    ShoppingBasket sb = (ShoppingBasket) ctx.lookup(„ShoppingBasket“);  
  
    ... // do something  
}
```

3.0

Session Bean

- Keine create()-Methode mehr notwendig
 - ▶ Stateless Session Beans besitzen kein Home Interface
 - ▶ create()-Methode(n) gibt es nicht mehr
 - **SLSB: create()-Methode war nicht sinnvoll**
 - **SFSB: Initialisierung der Bean erfolgt durch Aufruf entsprechender Geschäftsmethode(n)**
 - Keine explizite Annotation existent
 - `@Init` dient der EJB 2.x Migration

- Business-Interface muss vorhanden sein
 - ▶ Einfaches Java-Interface (POJI)
 - ▶ Kein Erben von `EJBObject` oder `EJBLocalObject`
 - ▶ ...damit auch keine `RemoteExceptions` mehr in den Methoden
 - ▶ Business-Interface kann generiert werden

Session Bean

- Bean-Klasse muss nicht mehr das Callback-Interface `javax.ejb.SessionBean` implementieren
 - ▶ `ejbRemove()` gibt es nicht mehr für SLSB
 - war für eine Stateless Session Bean eine überflüssige Methode
 - ▶ `Remove()`-Methode kann bei SFSB beliebige Methode der Bean-Klasse sein
 - Annotiert durch `@Remove`



Session Bean

■ Beispiel

Business Interface

```
@Remote public interface ProductManager {  
    public String getProductDescription(int prodID);  
}
```

Bean-Klasse

```
@Stateless public class ProductManagerBean implements ProductManager {  
    public String getProductDescription(int productID) {  
        ...  
    }  
}
```



Agenda

EJB 2.x und modellgetriebene Entwicklung

"Zurück" zu POJO/POJI

Verwendung von Meta-Annotationen

Kapselung von Umgebungs- und JNDI-Zugriffen

Weitere Neuerungen / Erleichterungen

Session Bean (Stateful)

Message-driven Bean

Persistent Entity / Persistenz

Fazit



Message-driven Bean

- `@MessageDriven` annotiert Bean-Klasse als Message-driven Bean
- Dependency Injection nutzbar
- Lifecycle Callbacks nutzbar
- Interceptor-Mechanismus nutzbar
- ...ansonsten hat sich nicht so viel getan ;-)



Agenda

EJB 2.x und modellgetriebene Entwicklung

"Zurück" zu POJO/POJI

Verwendung von Meta-Annotationen

Kapselung von Umgebungs- und JNDI-Zugriffen

Weitere Neuerungen / Erleichterungen

Session Bean (Stateful)

Message-driven Bean

Persistent Entity / Persistenz

Fazit



Persistent Entity / Persistenz

- Ziel: Vereinfachung/Redesign von Entity Beans → POJOs
 - ▶ Leichtgewichtige Domänen-Objekte

- Weiterentwicklung Persistenz: ursprüngliche Ziele

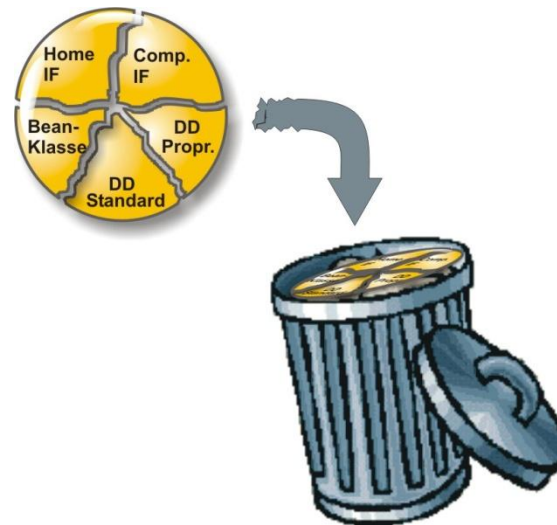
- ▶ CMP/BMP Nutzung vereinfachen
- ▶ Leistungsumfang erhöhen
- ▶ Schwerpunkt CMP und EJB-QL

- Ergebnis: Vollständiger Neuentwurf


- ▶ Massiv beeinflusst von

- Hibernate
- JDO
- TopLink

Best of Breed



Persistent Entity / Persistenz

- Entity Beans wie früher (vor EJB 3.0) gibt es nicht mehr
 - ▶ CMP und BPM in der alten Form in EJB 3.0 nicht mehr existent
 - Persistent Entity
 - ▶ EINE gemeinsame Persistence API für Java EE und Java SE
 - ▶ Entity Bean nicht mehr sinnvoll → Persistent Entity
 - Java Persistence Query Language ersetzt/erweitert EJB QL
- 
- `@Entity` annotiert POJO als Persistent Entity

Persistent Entity / Persistenz

- Home Interface entfällt
- Kein Business Interface mehr nötig (technisch betrachtet ;-)
- Eliminieren der komplexen, aufwendigen und fehleranfälligen Deployment Deskriptoren (optional nutzbar)
 - **Beziehungen etc. werden in den Klassen per Annotationen deklariert**
 - @OneToOne
 - @OneToMany
 - @ManyToOne
 - @ManyToMany

Persistenz Entity / Persistenz

- DTO/VO für Transport von Daten nicht mehr notwendig
 - ▶ Persistent Entity selbst wird transferiert (detached object)
 - ...implements Serializable
 - ▶ Nach Änderungen am 'detached object' können diese wieder mit persistentem Zustand zusammengebracht werden (reattached object)
 - **Verantwortlich für das Zusammenführen (mergen) der Daten ist Entity Manager**
 - ▶ Vereinfachung der Struktur der Applikationen
- Persistent Entities beherrschen nun Vererbung und Polymorphie
 - ▶ Sind damit voll objektorientiert

Persistent Entity / Persistenz

- Remotezugriff auf Persistent Entities nicht mehr direkt möglich!
- Zugriff auf Persistent Entities über `javax.ejb.EntityManager`
- EntityManager
 - ▶ Erzeugen, löschen, speichern von persistenten Entity-Instanzen
 - ▶ Finden von Entitäten über Primärschlüssel
 - ▶ Abfragen über Entitäten

```
@Stateless public class OrderEntryManagerBean
implements OrderEntryManager {
    @PersistenceContext private EntityManager em;

    public int createOrder(String articleNo, int quantity, ...) {
        Order order = new Order(articleNo, quantity, ...);
        em.persist(order);
        return order.getId();
    }

    public Order find(int id) {
        return em.find(Order.class, id);
    }

    public void enterOrder(int custID, Order newOrder) {
        Customer cust = (Customer)em.find("Customer", custID);
        cust.getOrders().add(newOrder);
        newOrder.setCustomer(cust);
    }
}
```

Persistenz Entity / Persistenz

■ Fazit

- ▶ EJB 3 bietet eine Menge Möglichkeiten, die einer MDSD-Verwendung entgegenkommen
 - **Deployment-Deskriptoren nicht mehr zwangsweise nötig**
 - **Schnittstellen und Callbackinterfaces sind entfallen → Vereinfachung Mikroarchitektur**
 - EntityBean, MessageDrivenBean, SessionBean entfallen beispielsweise
 - Home Interfaces, Component Interfaces entfallen
 - Methoden, die implizit (also ohne Interfacevorgabe) vorhanden sein müssen entfallen
 - **POJOs und POJIs**
 - **Technische Aspekte verschwinden aus dem Code (keine RMI-abhängigen Exceptions etc. mehr)**
 - **Persistent Enties sind echt objektorientiert und lassen auch Vererbung und Polymorphie zu**
- ▶ Es werden sich leichter MDSD-Tools dafür entwickeln lassen (Analog Spring etc.)
- ▶ Modellierung mit EJB 3 ist „sauber“, objektorientiert

Fragen & Antworten Diskussion

oliver.ihns@holisticon.de
www.holisticon.de

