EJB 3 - Erfahrungen aus der Praxis

Heiko W. Rupp
heiko.rupp@redhat.com

Agenda

- Background
- Kurzabriss Standard
- Erfahrungen aus der Praxis
  - Umstieg von EJB 2
  - Neuentwicklung

Agenda

- Background
- Kurzabriss Standard
- Erfahrungen aus der Praxis
  - Umstieg von EJB 2
  - Neuentwicklung

Background

- Service Level Management System JBossON
  - Kern übernommen
  - Entwickelt in EJB 2.x
  - Diverse Probleme in der Weiterentwicklung

- Entscheidung zur Weiterentwicklung in EJB 3
  - Umstieg über Tools
    - Generierung von Entity Beans via NetBeans
    - Umwandlung der EJB-2-Finder über Perl-Skript
  - Neuentwicklung nach Änderung des Datenmodells

- Ziel: (fast) die ganze Zeit ein lauffähiges System

Background (2)

- Domain-Modell
  - 88 Tabellen
  - 99 Entity Beans

- Geschäftslogik
  - 33 Stateless Session Beans

- GUI
  - Alt: Struts
  - Neu: Verlagerung hin zu JSF

Agenda

- Background
- Kurzabriss Standard
- Erfahrungen aus der Praxis
  - Umstieg von EJB 2
  - Neuentwicklung

Abriss des Standards

Abriss des Standards

- Ziele
  - Zurück zum POJO
    - Keine komplexen Interfaces
  - Konvention statt Konfiguration
    - Sinnvolle Voreinstellungen
- Metadaten am Code (möglich)
  - Weniger Artefakte notwendig

Abriss des Standards

- Ziele
  - Zurück zum POJO
    - Keine komplexen Interfaces
  - Konvention statt Konfiguration
    - Sinnvolle Voreinstellungen
  - Metadaten am Code (möglich)
    - Weniger Artefakte notwendig
- Enterprise Java Beans
  - Session Beans und Message-Driven Beans „wie seither“
    - Aber einfacher
  - Entity Beans neu als Java Persistence Api (JPA)
    - Sehr ähnlich Hibernate / JDO
    - Layer über Persistenz-Provider
    - Ansprache über EntityManager

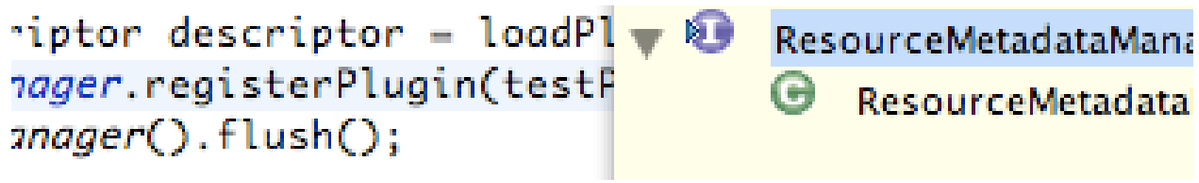
Konsequenzen aus der Erneuerung

Konsequenzen aus der Erneuerung

- Schnellere Entwicklung
  - Weniger Code
  - Kein synchron-halten vieler Artefakte
  - Direkte Aufruf-Verfolgung im Editor

Konsequenzen aus der Erneuerung

- Schnellere Entwicklung
  - Weniger Code
  - Kein synchron-halten vieler Artefakte
  - Direkte Aufruf-Verfolgung im Editor



Konsequenzen aus der Erneuerung

- Schnellere Entwicklung
  - Weniger Code
  - Kein synchron-halten vieler Artefakte
  - Direkte Aufruf-Verfolgung im Editor
- Schnelleres und besseres Testen
  - Entity Beans in Unit Tests
  - Dank JBoss Embedded EJB 3: Session Beans in Unit Tests !
    - Inklusive Dependency Injection
- Volle Funktionalität kann im CruiseControl (etc.) getestet werden

Konsequenzen aus der Erneuerung

- Schnellere Entwicklung
  - Weniger Code
  - Kein synchron-halten vieler Artefakte
  - Direkte Aufruf-Verfolgung im Editor
- Schnelleres und besseres Testen
  - Entity Beans in Unit Tests
  - Dank JBoss Embedded EJB 3: Session Beans in Unit Tests !
    - Inklusive Dependency Injection
- Volle Funktionalität kann im CruiseControl (etc.) getestet werden

Test-Domain	Anzahl Tests	Laufzeit (*)
Entities	57	25 s
Geschäftslogik	105	3 min

(*) Inkl. Maven Overhead und Sleep()

Agenda

- Background
- Kurzabriss Standard
- Erfahrungen aus der Praxis
  - Umstieg von EJB 2
  - Neuentwicklung

Naming

- Dependency Injection statt explizitem JNDI-Lookup
  - Field-Injection

```
@EJB
SessionBeanLocal mySB;
```

  - Setter-Injection

```
SessionBeanLocal mySB;

@EJB public void setSessionBeanLocal
  (SessionBeanLocal local) {mySB = local;}
```

  - Reine Lehre sagt „Setter-Injection“
  - Praxis sagt „Field-Injection“
- Namen im JNDI leider weiterhin herstellerspezifisch
  - Teils anders für App. im JAR / EAR

Java Persistence API (JPA)

- Im ersten Durchgang
  - Generierung der Entitys durch NetBeans aus DB Schema
  - Funktioniert soweit gut
  - Oft wurden die falschen Java-Datentypen verwendet
    - Z.B. Postgres int4 -> long
  - Datentabellen kennen keine Vererbung
    - Damit auch kein Klassenmodell mit Vererbung
  - Namen für Relationen waren „seltsam“
    - Viel händische Nacharbeit
    - Kein einfaches Ersetzen der VO durch Entitys
- Überlegung:
  - Umwandeln der EJB 2.x CMP Entity-Beans via Perl-Script

PersistenceContext/-Unit

- PersistenceUnit
  - Sammlung von Entity Beans
- PersistenceContext
  - Wie Hibernate-Session
  - Von EntityManager verwaltet

```
@PersitenceContext
EntityManager em;
```

- Einzig notwendiger Deployment-Deskriptor:
  persistence.xml
- Szenario:
  - Entity Beans in einem Archiv
  - Nutzende Session Beans in anderem Archiv
- @PeristenceContext braucht Namen aus persistence.xml

```
@PersistenceContext(unitName = „myPU“)
EntityManager em;
```

EJB-Finder / JPA-Query

EJB-Finder / JPA-Query

- In EJB 2 am Home-Interface
  - „statische Methode“ + EJB-QL Abfrage im Deskriptor
  - Namenscheck zur Compile-Zeit
  - Sehr limitierte Abfragesprache

EJB-Finder / JPA-Query

- In EJB 2 am Home-Interface
  - „statische Methode“ + EJB-QL Abfrage im Deskriptor
  - Namenscheck zur Compile-Zeit
  - Sehr limitierte Abfragesprache
- In EJB 3
  - Methode in Session Bean notwendig
  - Mächtige Abfragesprache
  - Verwendung von SQL vorgesehen
  - Ad-hoc Querys („Stringbastelei“)
  - Deklaration am Entity Bean möglich --> „Named Query“
    - Syntaxüberprüfung zur Deploy-Zeit bzw. bei Unit-Tests
    - Container kann sie vorkompilieren

Named Querys

- Name muss innerhalb der PersistenceUnit eindeutig sein
  - Schwierig bei vielen Entitys (z.B. „findByName“)

Named Querys

- Name muss innerhalb der PersistenceUnit eindeutig sein
  - Schwierig bei vielen Entitys (z.B. „findByName“)
  - Voranstellen AbstractSchema

```
@NamedQuery( name=“Kunde.findByX“, query=“...“)
```

  - Nutzung:

```
Query q = em.createNamedQuery(„Kunde.findByX“);
```

Named Querys

- Name muss innerhalb der PersistenceUnit eindeutig sein
  - Schwierig bei vielen Entitys (z.B. „findByName“)
  - Voranstellen AbstractSchema

```
@NamedQuery( name=“Kunde.findByX“, query=“...“)
```

  - Nutzung:

```
Query q = em.createNamedQuery(„Kunde.findByX“);
```

  - Besser als Konstante

```
@NamedQuery(name=Kunde.FIND_BY_X, ...)
@Entity
public class Kunde
{
   public static final String
   FIND_BY_X=“Kunde.FIND_BY_NAME“;
```

  - Weniger Fehler zur Laufzeit
  - Leichteres Auffinden der Query im Code

Named Querys / JPA-QL

- Mit den Fähigkeiten wachsen die Ansprüche
- Limitationen (einige ...)
  - Mandantenfähigkeit über mehrere Tabellen schwierig
    - Abstract Schema kann nicht auf unterschiedliche DB-Schemata zeigen
  - Viele Named Querys bei Sortierung nach unterschiedlichen Spalten
    - Reihenfolge bei ORDER BY nicht parametrierbar
    - Je eine Query für ASC und DESC
- Unsere Lösung:
  - @NamedQuery Annotation auslesen
  - Sortierung anfügen
  - Als normale Query ausführen
- Vorteile der Lösung:
  - Normale Query trotzdem durch Provider überprüfbar
  - Reduktion der Redundanz

PersistenceContext Teil 2

- 1st-Level-Cache
  - Identitätsgarantie
  - Tracking von Änderungen
  - Achtung bei equals() und hashCode() !
- Management des Lebenszyklus
  - Detached Objects
  - Entity Beans können zu Clients gesendet werden
    - VO / DTO müssen nicht mehr sein
    - Re-attach im Entity Manager
- VO eher für Multi-Entity-Querys
- Update in Batches

PersistenceContext als Cache

- Cachen der Beans bis insert / update
  - viele Beans: Flush nötig
- Querys schreiben den Cache raus
  - Antipattern:

```
for (int i=0; i< 100000; i++) {
   XY xy = new XY(i);
   em.perist(xy);
   Query q = ...;
   Object o = q.getSingleResult();
   xy.setZ(o);
}
```

PersistenceContext und Batch-Updates

- Endlich Batch-Update und -Delete
- Laufen am PersistenceContext vorbei
- Reihenfolge der Operationen ist wichtig

```
public void foo() {
   Query q = ... // Lese Kunden
   List<Kunde> kunden = q.getResultList();
   q = ... // Delete K from Kunde where k.Umsatz = 0
   q.executeUpdate();
   for (Kunde k: kunden) {
      // machwas mit dem Kunde
      // Problem: Kunden ohne Umsatz sind nicht mehr in der DB
   }
```

Annotationen für Entity Beans

- Ort von @Id bestimmt Zugriff durch EntityManager
  - Feld / Getter
  - Muss für die Hierarchie gleich sein
- Tools erwarten eher @Id am Feld
  - Dann keine Getter / Setter für den EntityManager notwendig
- „Reine Lehre“
  - @Id an Getter/Setter für Tests

- Realität:
  - Hibernate EntityManager / Embedded EJB3 für Tests
  - Feld-Zugriff möglich

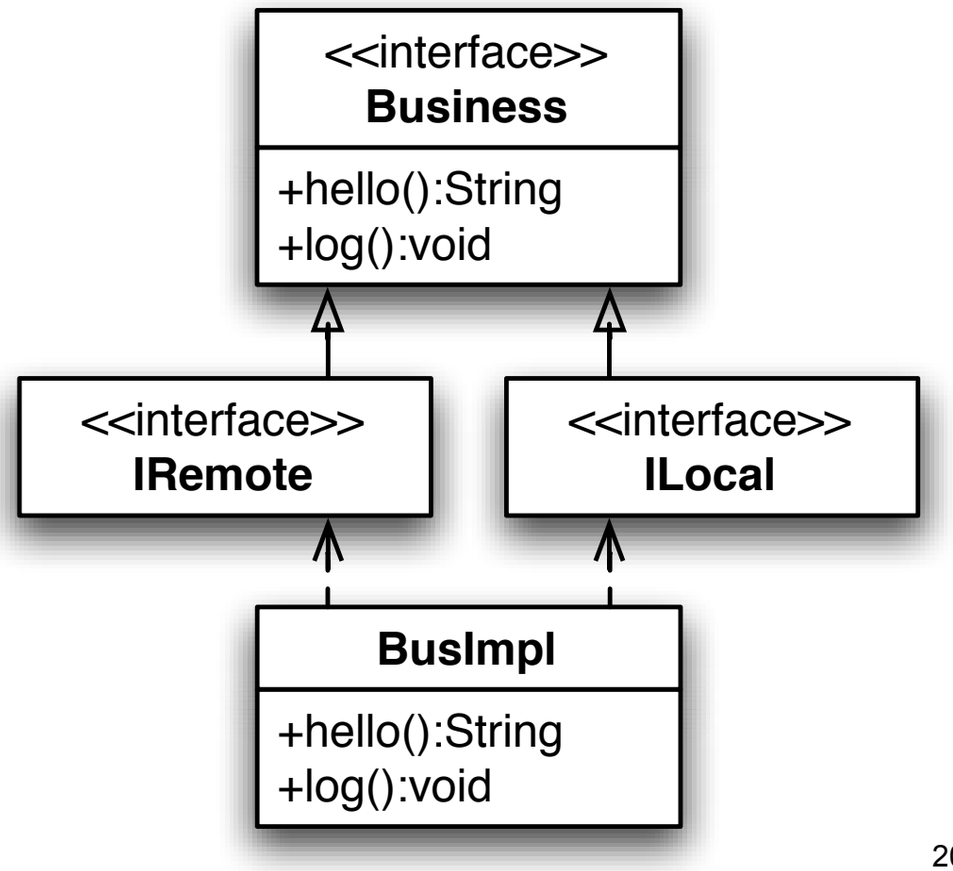
Session Bean

- @Local als Default (in 2.x war Remote der Default!)
- Kann nicht @Local und @Remote sein
- Muss dann Local- und Remote-Interface implementieren
- Auch SB als Webservice Endpoint benötigt ein Business-Interface

Session Bean

- @Local als Default (in 2.x war Remote der Default!)
- Kann nicht @Local und @Remote sein
- Muss dann Local- und Remote-Interface implementieren
- Auch SB als Webservice Endpoint benötigt ein Business-Interface

```
@Remote
public interface IRemote
   extends Business
{
}
```



Session Beans - Vererbung

- Session Beans können von anderen Session Beans erben
- Nicht offiziell in der Spezifikation
- Abschnitte über Security / Tx / Interceptoren in der Spezifikation
  - Haben entsprechende Kommentare
- Beispiel:
  - Session Bean „international“
  - Kind: Session Bean pro Land

Session Beans -- Transaktionen

- Transaktionen wie bisher
  - Einstellung über Annotation möglich
- Vereinfachtes Modell verleitet zu Fehlern

```
@Stateless
public class Foo implements FooIf {

    @EJB FooIf fooEjb;

    public void bar(){

          baz();

          fooEjb.baz();

    }


    @TransactionAttribute (REQUIRES_NEW)
    public void baz()   {

   }
```

Session Beans -- Transaktionen

- Transaktionen wie bisher
  - Einstellung über Annotation möglich
- Vereinfachtes Modell verleitet zu Fehlern

```
@Stateless
public class Foo implements FooIf {

    @EJB FooIf fooEjb;

    public void bar(){

          baz();          // tut nicht wie erwartet

          fooEjb.baz();   // So geht es

    }


    @TransactionAttribute (REQUIRES_NEW)
    public void baz()   {

   }
```

Session Beans -- Interceptoren

- Interceptoren in EJB 3.0
  - AOP Lite (nur around-invoke)
- Problem:
  - Java EE Security für uns unbrauchbar
- Lösung:
  - Eigene Annotation @RequiredPermission(PermisionType)
  - Subject immer als erster Parameter der Methode
  - Interceptor überprüft bei jedem Aufruf Annotation an Geschäftsmethode

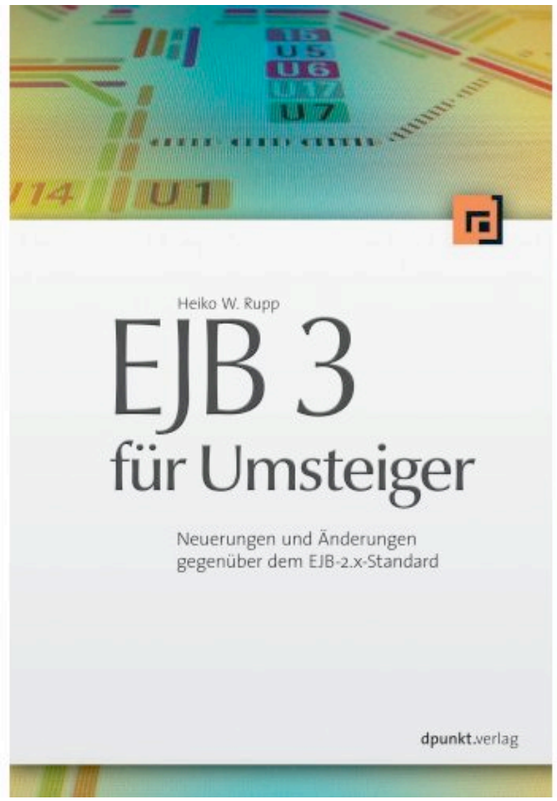
```
@Stateless public class Foo implements FooIF {

   @RequiredPermissions(PermissionType.MANAGE)
   public void bar(Subject user, String ... ) {  ... }
```

- Sauberes Herausziehen dieses Cross-Cutting-Concerns auf Standard-Art und Weise

Danke fürs Zuhören

- **Noch Fragen ?**

- Folien auf den Webseiten
- EJB 3 Beispielapplikation (Weblog-Syster
  - http://bsd.de/e3fu/
- JBossON:
  - http://network.jboss.com/



Mehr EJB 3 auf dem JFS 2007

- Nachfolgende Sessions:

- G4 (12:15)
  - „EJB 3.0 - Unit Testing Reloaded“ (Werner Eberling)
- D6 (15:35)
  - „Ausgemustert? Der Einfluss von EJB 3.0 auf Java EE Design Patterns (Stefan Heldt)