„Einführung in Hibernate und Demonstration“

Java Arbeitskreis der GTUG

20.11.2006

Jürgen Depping

Info@CommitWork.de
www.CommitWork.de

Agenda

- Motivation OR-Mapper
- Hibernate ein O/R-Mapper
- Hibernate auf HP-NonStop Systemen
- Hibernate Einführung mit Demo
- Hibernate Details
- AndroMDA und Hibernate

Motivation - OR-Mapper

Motivation - OR-Mapper

- Das objekt-orientierte Modell enthält reichere Strukturierungsmechanismen als das relationale Modell.

- Die folgenden Konzepte müssen transformiert werden:
  - Klassen
  - Aggregation
  - Assoziationen
  - Komposition
  - Vererbung

- Realisierungen:
  - manuelle Umsetzung mit JDBC
  - EJB mit Entity Beans
  - OR-Mapper
  - Objekt-orientierte Datenbanken

Motivation OR-Mapper

- manuelle Umsetzung in großen Projekten nur mit Hilfe von eigenen Generatoren sinnvoll.
  (CommitWork: DBGenerator – Klassenabbildung, Transferobjekt).

- EJB mit Entity Beans
  Vorteil: Standardisiert
  Nachteil: Umsetzung der OO-Konzepte nur mäßig

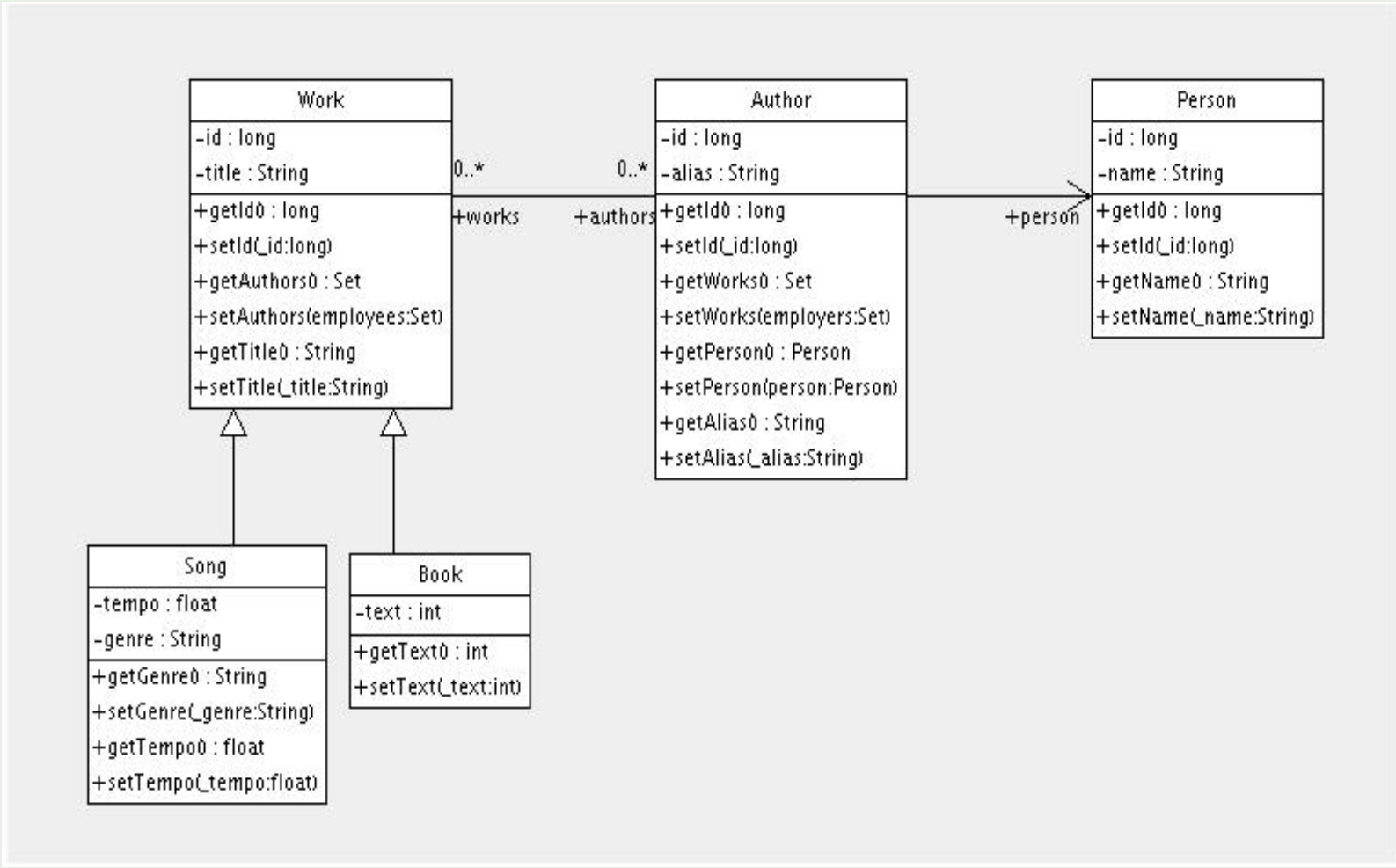
- Objekt-Orientierte Datenbanken:
  Vorteil: beste Umsetzung der Konzepte
  Nachteil: keine Standardisierung, keine Verbreitung, nachlassendes Interesse bei den Kunden

- OR-Mapper z.Zt. besser als Entity Beans, aber schlechter als OO-DB.

- Es gibt unterschiedliche OR-Mapper:
  IntelliBO der Fa. SignSoft, Hibernate Open Source, ...

Hibernatebeispiel

- Beispiel aus der Hibernatedokumentation:



Hibernate - ein OR-Mapper

Hibernate ein OR-Mapper

- Open Source Persistenz Framework für Java und C#
  - Die C# Version heißt NHibernate (LGPL - Lizenz)
  - ca. 30.000 Downloads pro Monat
  - Einsatz auch im kommerziellen Bereich (Lufthansa, Bank-Verlag,...)
- Kann auch im Cluster betrieben werden
- Vielfältig konfigurierbar:
  - Unterstüzung vieler Datenbanken, Caching Systemen, Transaktionsmonitoren, ...
- Abfragen mit eigene Abfragesprache HQL oder Criteria-API möglich
- Hibernate wird oft innerhalb eines Webcontainers wie Tomcat oder innerhalb von J2EE Applikationsservern verwendet (Bea WLS, JBoss,...)
- JBoss nutzt Hibernate zur Realisierung der EJB3 Spezifikation

Hibernate auf HP-NonStop Systemen

Alu Norf Studie von CommitWork durchgeführt – Januar 2006

1) Standardzugriffe CRUD auf eine MP-Tabelle über SQL-MX
   Testtablemp: eine SQL/MP-Tabelle mit Alias in SQL/MX
   Transaktionsverhalten:
   - bean managed transaction
   - container managed transaction

2) Standardzugriffe CRUD auf eine MX-Tabelle
   Testtablemx: eine reine SQL/MX-Tabelle,
   Transaktionsverhalten:
   - bean managed transaction
   - container managed transaction

3) XA-Test
   Innerhalb einer Transaktion Zugriff auf Oracle und SQL/MX

Ergebnisse Alu Norf Studie – Januar 2006

geforderte Eigenschaft	*Ergebnis*
Nutzung des WLS Connection Pools und der WLS Datasource	erfüllt
Verwendung von prepared Statements	erfüllt
Standardzugriff auf SQLMP mit SQLMX Alias Tabelle: Testtablemp Transaktion: BMP	erfüllt
Standardzugriff auf SQLMP mit SQLMX Alias Tabelle: Testtablemp Transaktion: CMP	erfüllt
Standardzugriff auf SQLMX Tabelle: Testtablemx Transaktion: BMP	erfüllt
Standardzugriff auf SQLMX Tabelle: Testtablemx Transaktion: CMP	erfüllt
Paralleler Zugriff auf eine nicht NonStop-Datenbank Tabelle: Person (separate Transaktion)	erfüllt
verteilte Transaktionen (XA) NonStop und Oracle Datenbank	erfüllt

SQL/MX Erweiterungen Hibernate - Dialektklasse

- Ziel: Datenbankunabhängigkeit
- Dialektklasse beschreibt datenbankspezifische Eigenschaften
- existiert für eine Vielzahl an Datenbanken
- für SQL/MX ist die generische Dialektklasse nicht ausreichend.
- SQL/MX: SQLMXDialect.java
  Package: org.hibernate.dialect
- Mapping von Java Datentypen auf DB Datentypen
- Quelle:“Using Hibernate with NonStop SQL/MX“ by Frans Jongma, HP EMEA

Notwendige Erweiterungen Hibernate - Batching

- SQL/MX benötigt ein explizites Löschen der Batchingverwaltung
- Klasse: org.hibernate.jdbc.BatchingBatcher
- Methode: doExecuteBatch(PreparedStatement ps)
- Anpassung: Auskommentieren des Aufrufes clearBatch().
- Quelle:“Using Hibernate with NonStop SQL/MX“ by Frans Jongma, HP EMEA

Wrapperklassen des SQL/MX-Typ4-Treibers

- Ohne Anpassung des SQL/MX-Typ4-Treibers sind die Hibernatetools, insbesondere das „reverse engineering“ nicht lauffähig.

- Ursache ist das Fehlverhalten der Methode getTables(...).
  Der Originaltreiber hat zwar die korrekte Signatur, aber eine falsche Semantik, obwohl das Verhalten dieser Methode allgemein festgelegt ist.

- Lösung: Durch Vererbung und Delegation kann das Verhalten nahezu korrigiert werden. CommitWork bietet entsprechende Wrapperklassen an. Der Original-Treiber muss aus lizenzrechtlichen Gründen vom Kunden selbst gepatcht werden!

Hibernate Einführung mit Demo

Demo Hibernate und Tools

Anhand einer einfachen Tabelle werden vorgestellt:

- Hibernate Konfiguration

- Hibernate Mapping
  - XML Mapping-Datei
  - Java POJO-Klasse (getter- und setter-Methoden)

- Verwendung der JBoss-IDE Hibernate Tools (Eclipse Plugin)
  - Hibernate Console
  - Abfragesprache HQL
  - Reverse Engineering

- Serverseitig verwenden wir für die Demo JBoss auf HP-NonStop.

- Aber zunächst ein wenig Theorie ...

Hibernate – Konfigurationsdateien (*.cfg.xml)

- Mit der Hibernatekonfiguration werden bestimmt:

  - Datenbank-Dialekt (SQLMX, PostgreSQL, ...) und Datenbankverbindung

  - Transaktionshandling

  - Caching

  - Connection Pool

  - welche Mappings verwendet werden (Abbildung der Tabellen auf Java POJO-Klassen)

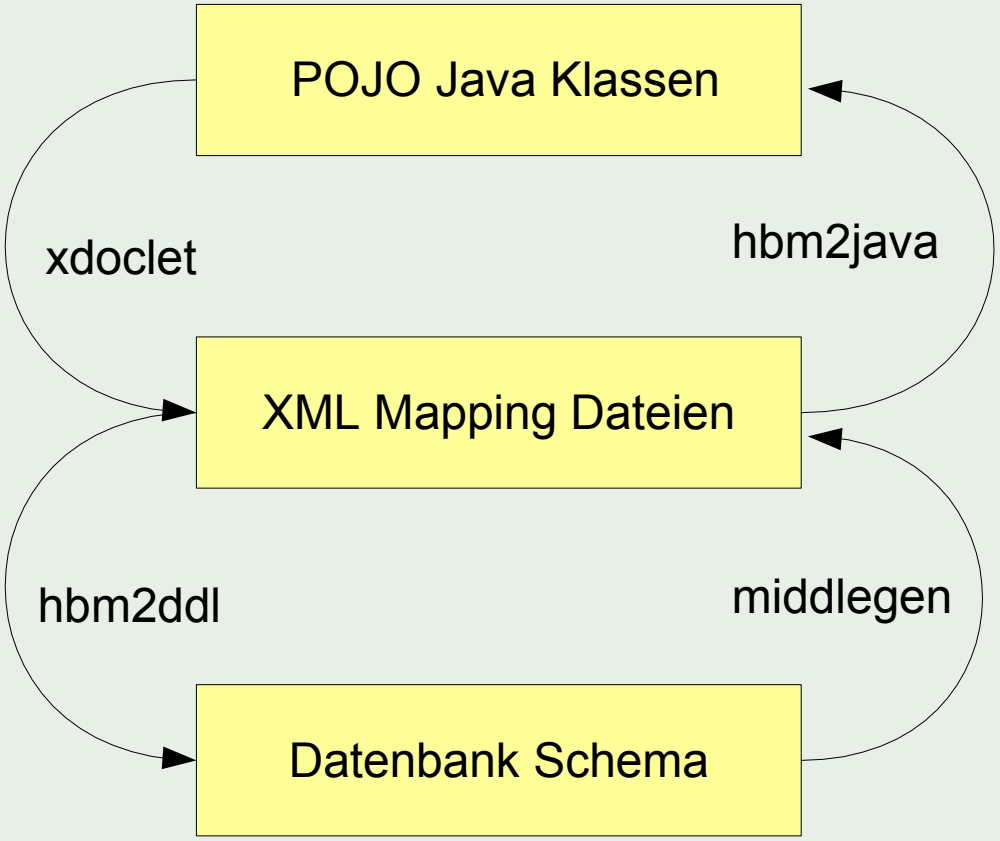
Hibernate – Mapping und POJO

- Mapping über XML (*.hbm.xml)
  - beschreibt eine Abbildung zwischen DB-Tabelle und Java-Klasse
  - enthält ggf. Zusatzinformation über Vererbung, Aggregation, Komposition und Assoziationen

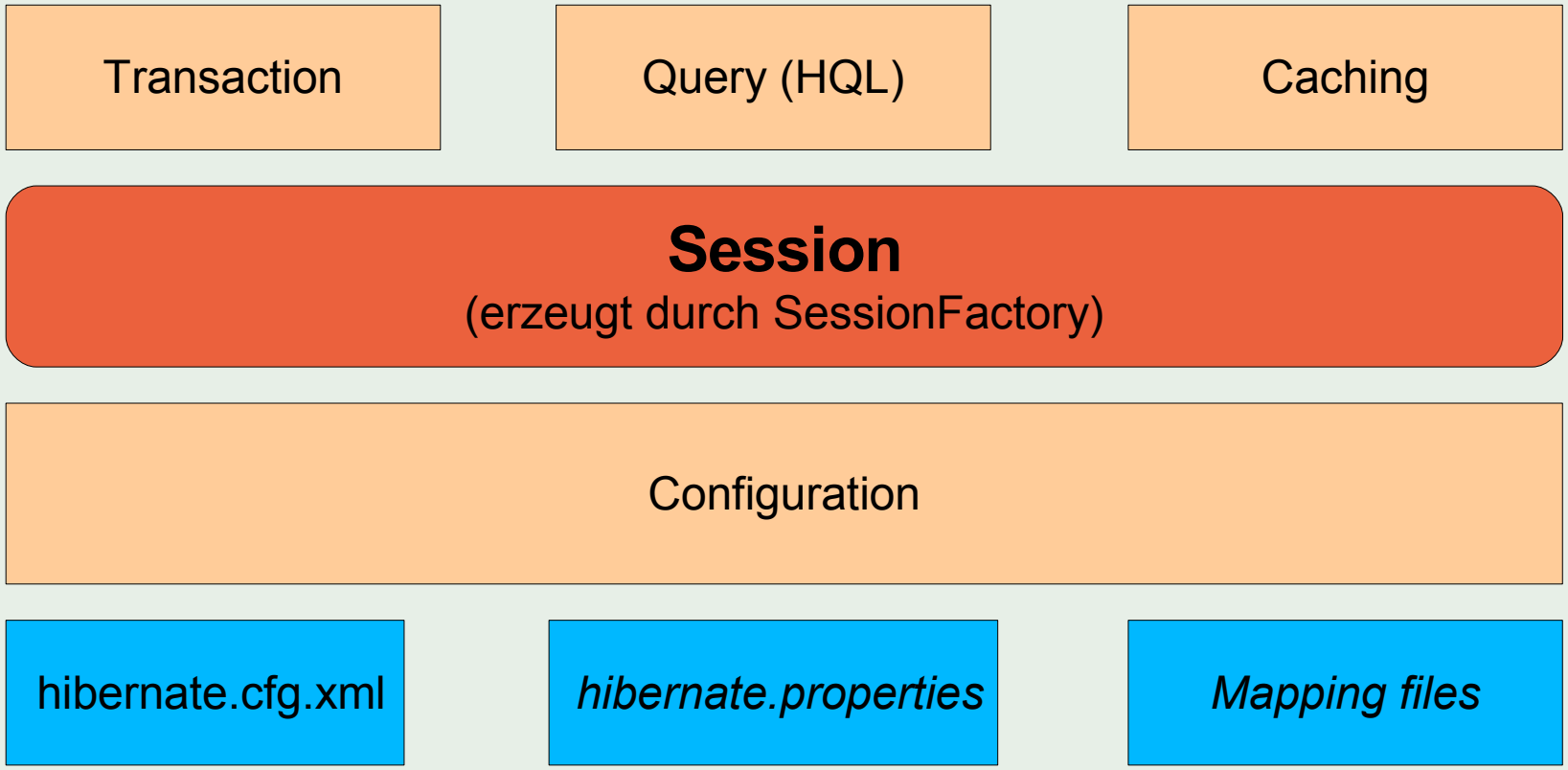
- POJO – Plain Old Java Object:
  - Java-Klasse, die die Datenbankobjekte enthalten
  - Jedes Attribut hat getter- und setter-Methoden
  - sind serialisierbar -> auf die Clientseite übertragbar

- Sowohl das Mapping (*.hbm.xml), als auch die POJO-Klasse können mittels „reverse engineering“ der Hibernatetools generiert werden!

Hibernate - Tools



Hibernate Komponenten



Hibernate – SessionFactory und Session

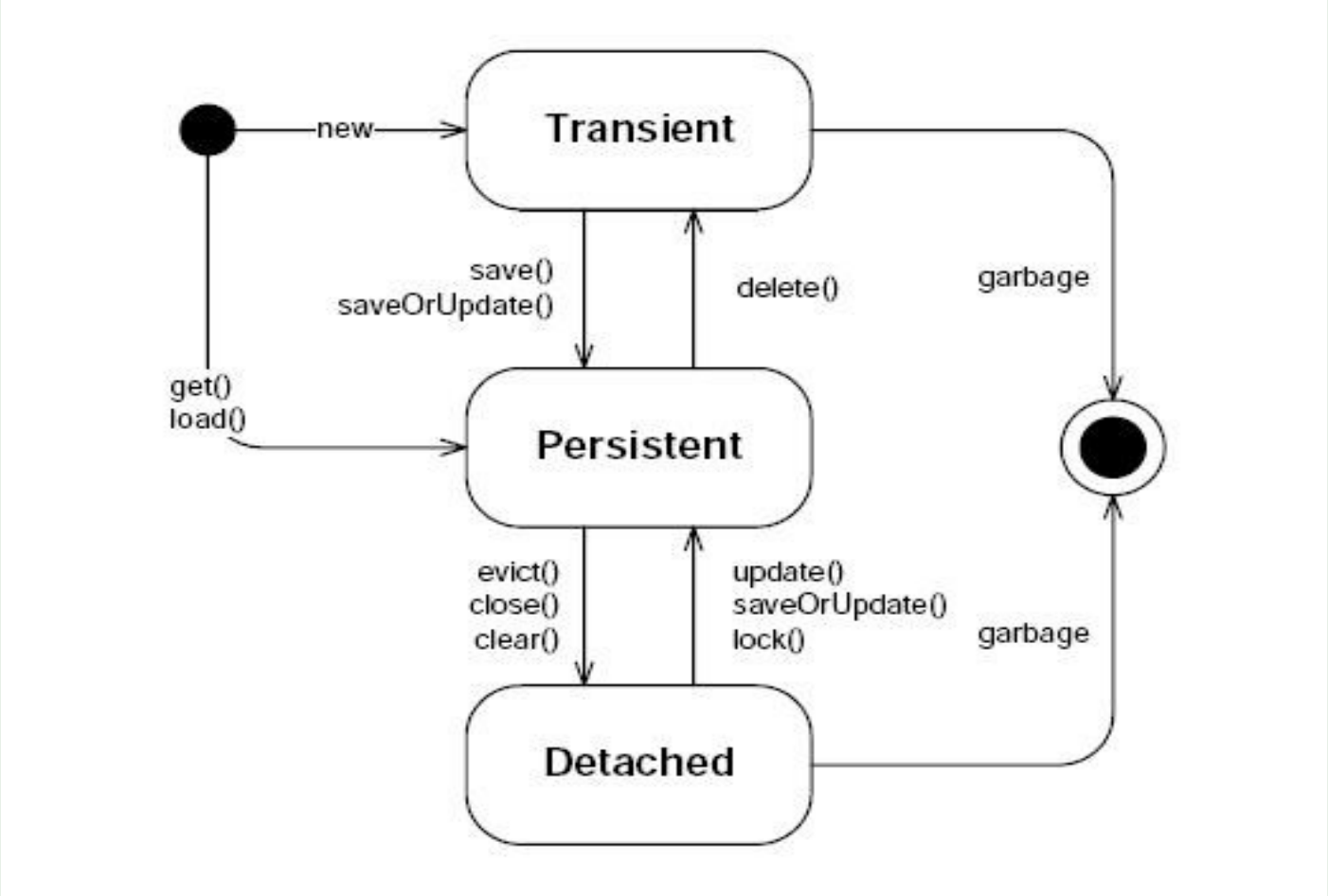
- SessionFactory erzeugt eine Session
- SessionFactory wird mit Hibernatekonfiguration konfiguriert:
  ...configure(„hibernateMeineKonfiguration.cfg.xml).buildSessionFactory;
- SessionFactory:

  - currentSession: liefert die aktuelle Session oder erzeugt eine Neue

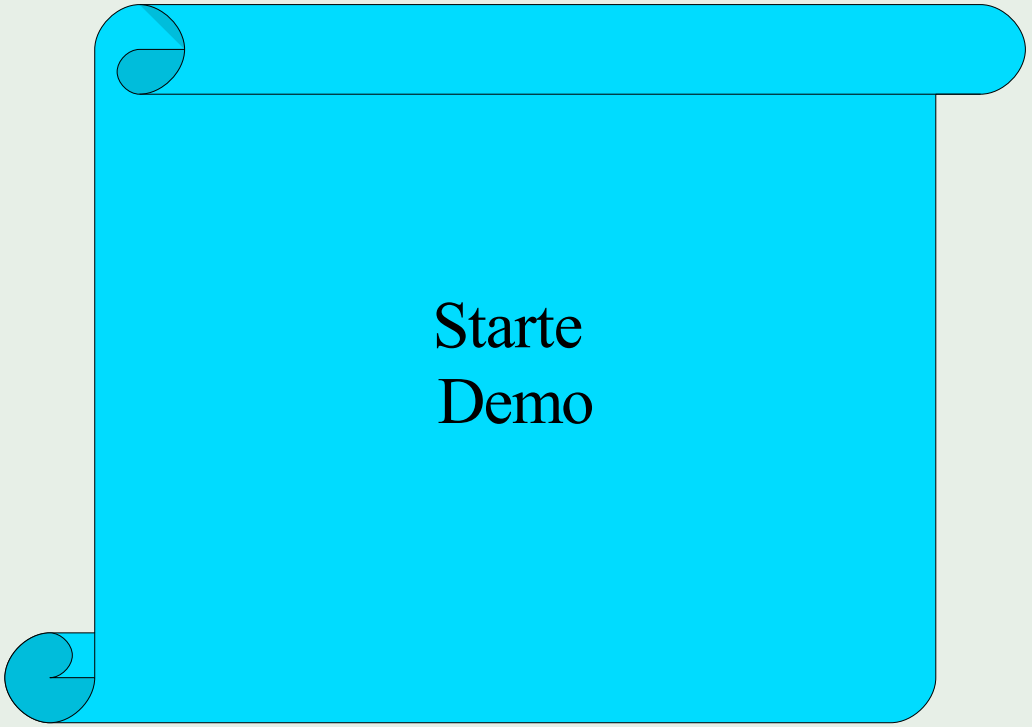
  - closeSession: schließt die aktuelle Session.

Hibernate Objektzustände

- Objektzustände: Transient, Persistent, Detached



DEMO



Hibernate Details

Hibernate Identifier

Primary keys (ids):

- Generierte (sequence, HILO, ...)

  HILO ist besonders für HP-NonStop interessant, da SQLMX keinen Mechanismus für generierte Keys besitzt!

- Von der Anwendung/Datenbank vergeben (assigned)

- Zusammengesetzte (composite-id)

Beispiel:

```
<id name="id" type="long">
          <column name="id" />
          <generator class="assigned" />
 </id>
```

Hibernate Locking

- Hibernate unterstützt selbstverständlich das **pessimistische Locking.**

- Interessant ist das **optimistische Locking.** Dabei reicht ein Eintrag in der Mapping Datei aus. Es ist keine zusätzliche Programmierung notwendig!

  Eingesetzt werden entweder ein Zähler oder ein Timestamp.

```
<version name="version" type="java.lang.Long">
        <column name="version" />
</version>
```

Fine-grained object model

- Hibernate bietet die Unterstützung des „fine-grained object model“ an.
- Für fine-grained Objekte werden keine separaten Datenbank Tabellen angelegt, sondern in der übergeordneten relationalen Tabelle enthalten.
- Diese Objekte existieren natürlich nur zusammen mit dem umgebenen Objekt.
- Innerhalb der Mapping Datei wird dies durch component beschrieben:

```
<class ...
   ...
   <component name="<Attributname>" class="<Klassenname>">
        <property .... />
   </component>
   ...
```

Assoziationen

- Assoziationen in einem Objektmodell:

  - unidirektionale und bidirektionale Assoziationen
    Navigation von einem Objekt zu einem oder einer Menge von Objekten

    Diese Navigation kann im Fall der bidirektionalen Assoziation von beiden Seiten erfolgen.

  - Hibernate unterstützt die Assoziationen:
    - **one-to-one**
    - **one-to-many** und **many-to-one**
    - **many-to-many**

  Bei Many-Assoziationen können unterschiedliche Java „Collection“-Klassen genutzt werden.

Kompositionen

- Eine Komposition gibt an, dass das Kindobjekt nur existiert, wenn das Vaterobjekt existiert.

  Also wird ein Kindobjekt automatisch gelöscht, wenn das Vaterobjekt gelöscht wird.

- Zur Abbildung einer Komposition wird in Hibernate der Zusatz <cascade="<Strategie>" /> verwendet.

  Cascading bietet unterschiedliche Stufen an.

Beispiel einer bidirektionalen One-to-Many Assoziation

```
<class
   name="com.commitwork.hibgen.transport.Customer"
   table="customer">
...
   <set  name="orders" ... inverse="true">
     <key column="cust"> </key>
     <one-to-many class="com.commitwork...Order"/>
   </set>
...
</class>
```

Lazy Assoziationen

- Eine lazy Assioziation fordert die Daten der Assoziation erst beim tatsächlichen Zugriff an (Proxy).

  Objekte werden also erst bei Bedarf ermittelt (Performance).

Inheritance - Vererbung

- Damit ein Objektmodell abbildbar ist muss auch die objektorientiert Vererbung möglich sein.

- Hibernate bietet zur Abbildung drei Möglichkeiten an:

  1) Tabelle für jede konkrete Klasse
     Tabelle enthält Vater und Kind

  2) Tabelle für jede Klassen Hierachie
     Alle Väter und Kinder werden in einer Tabelle gehalten,
     Ein Discriminator gibt an, um welche Klasse es sich handelt.

  3) Tabelle je Subklasse
     Für Vater und Kind wird jeweils in eine eigene Tabelle angelegt,
     Die Beziehungen werden durch foreign keys abgebildet.

Objekt - Caching

- Hibernate unterstützt unterschiedliche Caching Provider.
  EHCache, OSCache, SwarmCache,TreeCache

  In der Hibernatekonfiguration wird der Caching Provider festgelegt.

- Interessant ist der TreeCache.
  Ist auch in einer geclusterten Umgebung einsetzbar!

Das Caching kann dann je Klasse konfiguriert werden:

```
<class
   name="com.commitwork.test1.transport.Testtable"
   table="testtable" schema="public">
  <cache usage="transactional" />
   ...
```

Vorteile für HP NonStop (SQL/MX)

- **Die 4-K Grenze fällt!**
  Man kann über die Mapping Datei zwei Tabellen zusammenführen, so dass aus Programmiersicht die POJO-Klasse mehr als 4-k Daten enthält. Diese Problem konnten wir bei der Portierung von einer nicht NonStop Platform auf HP-NonStop nutzen!

- **Generierte Keys:**
  Automatisch generierte ID's haben wir durch Nutzung des HILO-Generators implementiert! Von der Datenbank generierte Keys werden zur noch nicht von SQL/MX unterstützt!

- **Perpared Statements:**
  Hibernate passt ideal zur HP-NonStop, da „prepared Statements“ verwendet werden!

Alternativen zur XML-Notation der Mappingdateien

- **Problem: XML**
  Viele Kunden möchten die Eigenschaften der Mapping Datei nicht in XML kodieren.

- **Lösung: Einsatz von XDoclet**
  Hierzu bietet Hibernate die Nutzung von Metakommentaren innerhalb der Java Pojo Datei an. Durch XDoclet werden aus diesen Meta-Kommentaren die notwendigen XML-Mapping Dateien erstellt.

- **Lösung: Nutzung eines UML-Tools mit MDA-Generators**
  Hier gibt es unterschiedliche Tools.
  z.B.: AndroMDA ein freier Generator für Model Driven Architecture.

  Eine Demonstration folgt!

Nutzung von Meta-Kommentaren und XDoclet

```
package com.commitwork.hibgen.transport;
/**
 * @hibernate.class table="order"
 */
public class Order implements Serializable {
...
/**
 * @hibernate.id column="id"
 * generator-class="native" not-null="true"
 */
...
public long getOrderId() {return orderId;}
...
```

AndroMDA und Hibernate

Model Driven Architecture

AndroMDA – Model Driven Architecture

- Model Driven Architecture ist schon seit einiger Zeit im Gespräch.

- Vorgehensweise:

  1. Erstellung des objektorientierten Modells mit UML.
     (UML=unified modeling language)

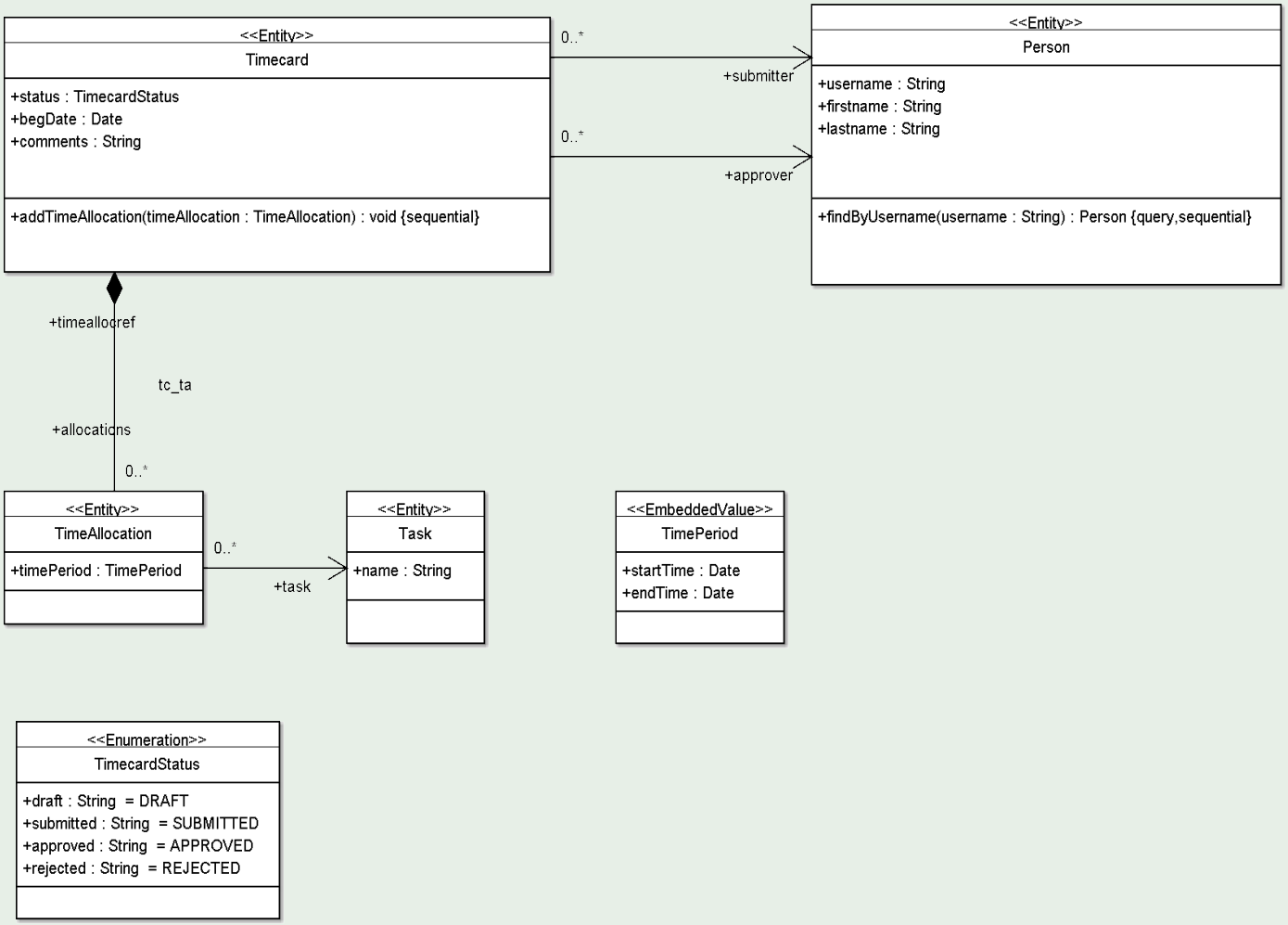
     Hier existiert mittlerweile eine Vielzahl an Produkten.
     Argo-UML (Open Source), Poseidon, Magic Draw, Rational Rose, ...

  2. Das Modell wird als XMI-Datei exportiert.

  3. Ein MDA Generator erstellt hieraus Programmquelltexte in der Zielsprache (C++, Java)

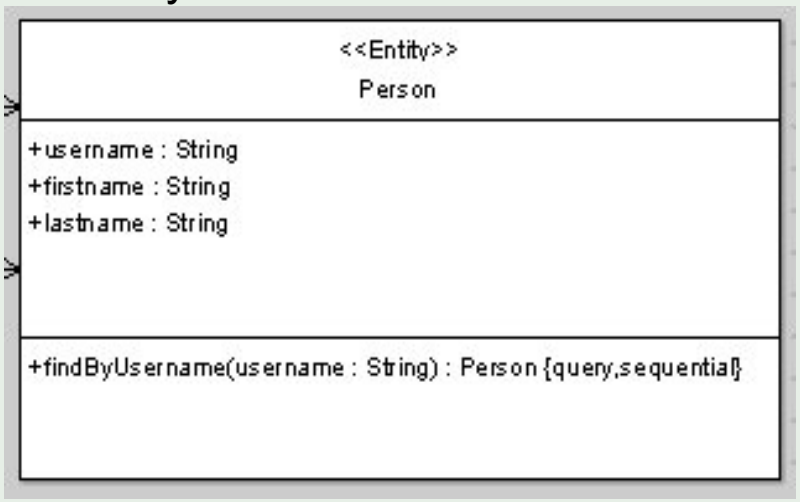
- Ein MDA-Generator ist AndroMDA (Open Source)
  Erstellt werden hier Java Quellcode und Hibernate Dateien!

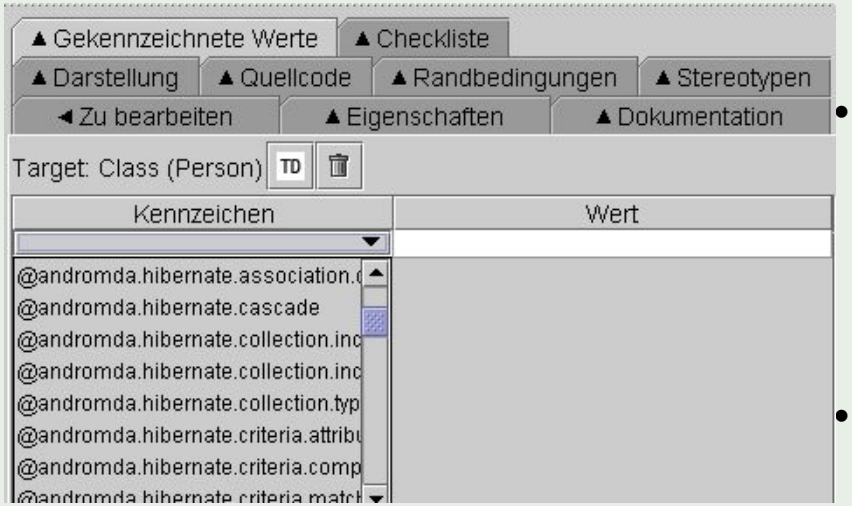
Ein UML Klassendiagramm – Beispiel AndroMDA



Die Entity Person

- Entity Person:





- <<Entity>> ist ein Stereotyp.

  Der Stereotyp teilt dem MDA-Generator mit, was erzeugt werden soll.

  Entity => Erzeuge eine Hibernate Mapping Datei und die zugehörige POJO Java Datei.

- Über die „gekennzeichneten Werte“ können zusätzliche Hibernate Eigenschaften gesetzt werden.

- Die Assoziaten wurden hier nicht dargestellt (Demo)!

AndroMDA DEMO

Fazit Model Driven Architecture

- Anmerkung: Zur Zeit wird ein Eclipse Plugin für AndroMDA entwickelt!

- MDA ist aus Sicht eines OO-Designers der Idealfall.

- Probleme:
  - Teamfähigkeit bei Nutzung eines UML Case Tools ist zu klären

  - Durch MDA-Einsatz besteht die Gefahr in eine Versions- und Tool-Hölle zu landen. Für das Gesamtprojekt werden viele Tools eingesetzt. Es muss sichergestellt werden, dass die Versionen passen. Vor allen Dingen bei einem Upgrade einer Komponente!

  - Arbeitsweise wird vorgeben:
    Erst das Modell verändern => generieren => implementieren

  - Case Tool Verhalten bei Großprojekten ist zu testen!
    Praktischer Lösungsansatz: nur das Domain Modell wird betrachtet!

Fazit

Fazit Hibernate

- Hibernate erfüllt die geforderten Eigenschaften eines O/R-Mappers.
- Sie haben innerhalb des Workshops einen Eindruck erhalten, was mit Hibernate heute möglich ist. Die Abfragesprache HQL wurde aus Zeitgründen nicht näher dargestellt.
- Sie wissen was Modell Driven Architecture bedeutet.
- Hibernate ist auf HP-NonStop Systemen einsetzbar (SQL/MX).
- Hibernate wird heute schon kommerziell eingesetzt.
- Die Zukunft wird zeigen, ob O/R-Mapper die reine JDBC-Programmierung ablösen werden.