Christian Kühl

Freie Universität Berlin

Von JDO zu JPA – Entwicklung einer Migrationsstrategie

Inhalt

- JDO und JPA
- Motivation
- Was muss migriert werden?
- Strategien
- Architektur
- Beispiel
- Migration am Beispiel
- Automatisierung
- Probleme
- Tests

JDO und JPA

- Spezifikationen zur Umsetzung von Persistenz mit Java
- JDO 1.0 standadisiert im Jahr 2001 und die aktuellste Version ist 3.0.1 (11.2011)
  - Provider: Kodo, Data Nucleus
  - Configuration over Convention
- JPA 1.0 standadisiert 2006, 2.0 2009 und aktuell 2.1 (05.2013)
  - Provider: Hibernate, EclipseLink, Open JPA
  - Convention over Configuration

Motivation

- Kodo:
  - Lizenzen
  - Support
  - Wartung
  - Java 7
- Allgemein:
  - Mehr Entwicklung
  - Java EE
  - Convention over Configuration
  - JPA ist der Standard

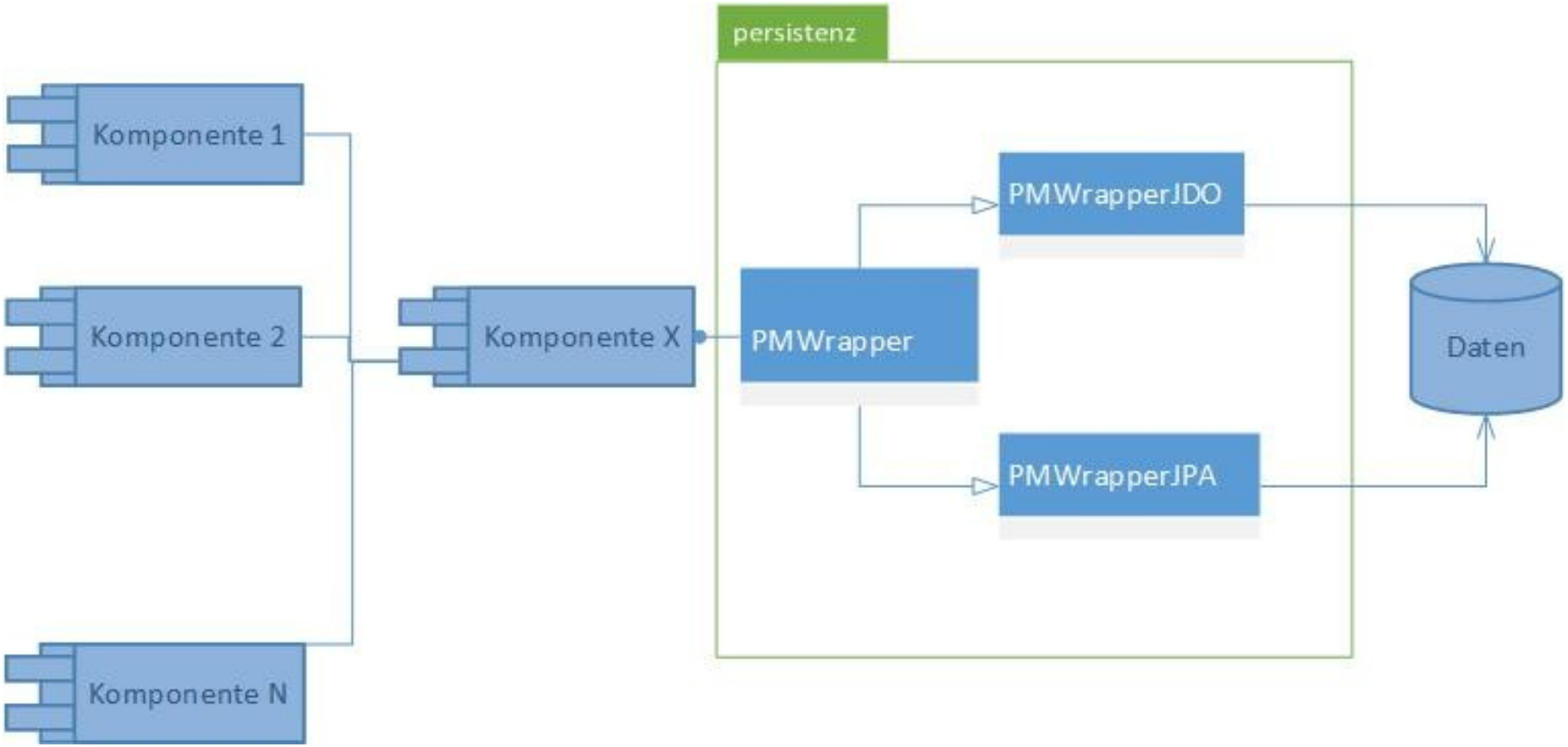
Was muss migriert werden?

- Konfiguration
  - Properties → XML
- Metadaten
  - XML → Annotations
- Mapping
  - XML → Annotations
- Query
  - JDOQL → JPQL

Strategien

- Komplett
  - Vorteile:
    - Wartungsaufwand bleibt gleich
    - Keine komplexere Architektur
  - Nachteile:
    - Kein Fallback
    - Alles muss migriert werden
- Hybrid 
  - Vorteile:
    - Fallback
    - Teilweise Migration möglich
  - Nachteile
    - erhöhter Wartungsaufwand
    - Architektur wird komplexer

Zielarchitektur - Hybrid



Beispiel

```
public class Person{
  private String name;
  private int age;
  private Address address;
...
}

public class Address{
  private String city;
  private int postalCode;
  private String street;
  private int housenumber;
...
}
```

Beispiel - Konfiguration

```
##Datenbank
javax.jdo.option.ConnectionDriverName: org.hsqldb.jdbcDriver
javax.jdo.option.ConnectionPassword:
javax.jdo.option.ConnectionURL: jdbc:hsqldb:mem:tmp
javax.jdo.option.ConnectionUserName: sa

##ID Generator
kodo.jdbc.SequenceFactory: db-class

##Liste der persistenten Klassen
kodo.PersistentClasses: package.Person, package.Address
```

Migration - Konfiguration

```
<persistence>
   <persistence-unit name="migration">
     <class>package.Person</class>
     <class>package.Address</class>
     <properties>
       <property name="javax.persistence.jdbc.driver"
                 value="org.hsqldb.jdbcDriver" />
       <property name="javax.persistence.jdbc.url"
                 value="jdbc:hsqldb:mem:tmp"/>
       <property name="javax.persistence.jdbc.user" value="sa"/>
       <property name="javax.persistence.jdbc.password" value=""/>
     </properties>
   </persistence-unit>
</persistence>
```

Beispiel - Metadaten

```
<jdo>
     <package name="package">
          <class name="Person" >
               <extension vendor-name="kodo" key="jdbc-class-ind-name" value="metadata-value"/>
               <extension vendor-name="kodo" key="jdbc-class-ind-value" value="442"/>
          </class>
          <class name="Address">
               <extension vendor-name="kodo" key="jdbc-class-ind-name" value="metadata-value"/>
               <extension vendor-name="kodo" key="jdbc-class-ind-value" value="441"/>
          </class>
     </package>
</jdo>
```

Migration - Metadaten

```
@Entity
@DiscriminatorValue(value=„442“)
public class Person{
private String name;
private int age;
private Address address;
...
}
```

```
@Entity
@DiscriminatorValue(value=„441“)
public class Address{
private String city;
private int postalCode;
private String street;
private int housenumber;
...
}
```

Beispiel - Mapping

```
<mapping>
    <package name="package">
        <class name="Person">
            <jdbc-class-map type="base" pk-column="JDOID" table="PERSON"/>
            <jdbc-version-ind type="version-number" column="JDOVERSION"/>
            <jdbc-class-ind type="metadata-value" column="JDOCLASS"/>
            <field name="name">
                <jdbc-field-map type="value" column="NAME"/>
            </field>
             <field name="age">
                <jdbc-field-map type="value" column="AGE"/>
            </field>
             <field name="address">
                <jdbc-field-map type="one-one" column.JDOID="ADDRESS_JDOID"/>
            </field>
        </class>
        ...
```

Migration Mapping

```
<jdbc-class-map type="base" pk-column="JDOID" table="PERSON"/>
```

```
@Entity
@DiscriminatorValue(value=„442“)
@Table(name=“PERSON“)
public class Person{
@GeneratedValue(generator=“PersonGenerator“ strategy = GenerationType.TABLE)
@TableGenerator(initialValue = 1000, name = "PersonGenerator", pkColumnName = "ID",
 table = "JDO_SEQUENCE", pkColumnValue = "package.Person",
valueColumnName = "SEQUENCE_VALUE")
@Column(name=“JDOID“)
@Id
private long jpaId;
private String name;
private int age;
private Address address;
...
```

Migration - Mapping

```
<jdbc-version-ind type="version-number" column="JDOVERSION"/>
<jdbc-class-ind type="metadata-value" column="JDOCLASS"/>
```

```
@Entity
@DiscriminatorValue(value=„4400002“)
@DiscriminatorColumn(name=“JDOCLASS“)
@Table(name=“PERSON“)
public class Person{
...
@Column(name=“JDOVERSION“)
@Version
Private int version;
private String name;
private int age;
private Address address;
...
}
```

Migration - Mapping

```
<field name="name"><jdbc-field-map type="value" column="NAME"/> </field>
<field name="age"><jdbc-field-map type="value" column="AGE"/> </field>
<field name="address"><jdbc-field-map type="one-one" column.JDOID="ADDRESS_JDOID"/>
      </field>
```

…

@Column(name=“NAME“)

private String name;

@Column(name=“AGE“)

private int age;

@JoinColumn(name = "ADDRESS_JDOID")

@OneToOne

private Address address;

...

}

Migration – Query

- Warum?
  - Unterschiede in Operatoren
    - == zu =
    - != zu <>
    - && zu AND
    - || zu OR
    - ! zu NOT
  - Anwendung Aggregierungs Funktionen
    - query.setResult(“count(this)”);
    - em.createQuery("SELECT COUNT(b). . .“)
  - Typsicherheit ->Criteria API

Beispiel

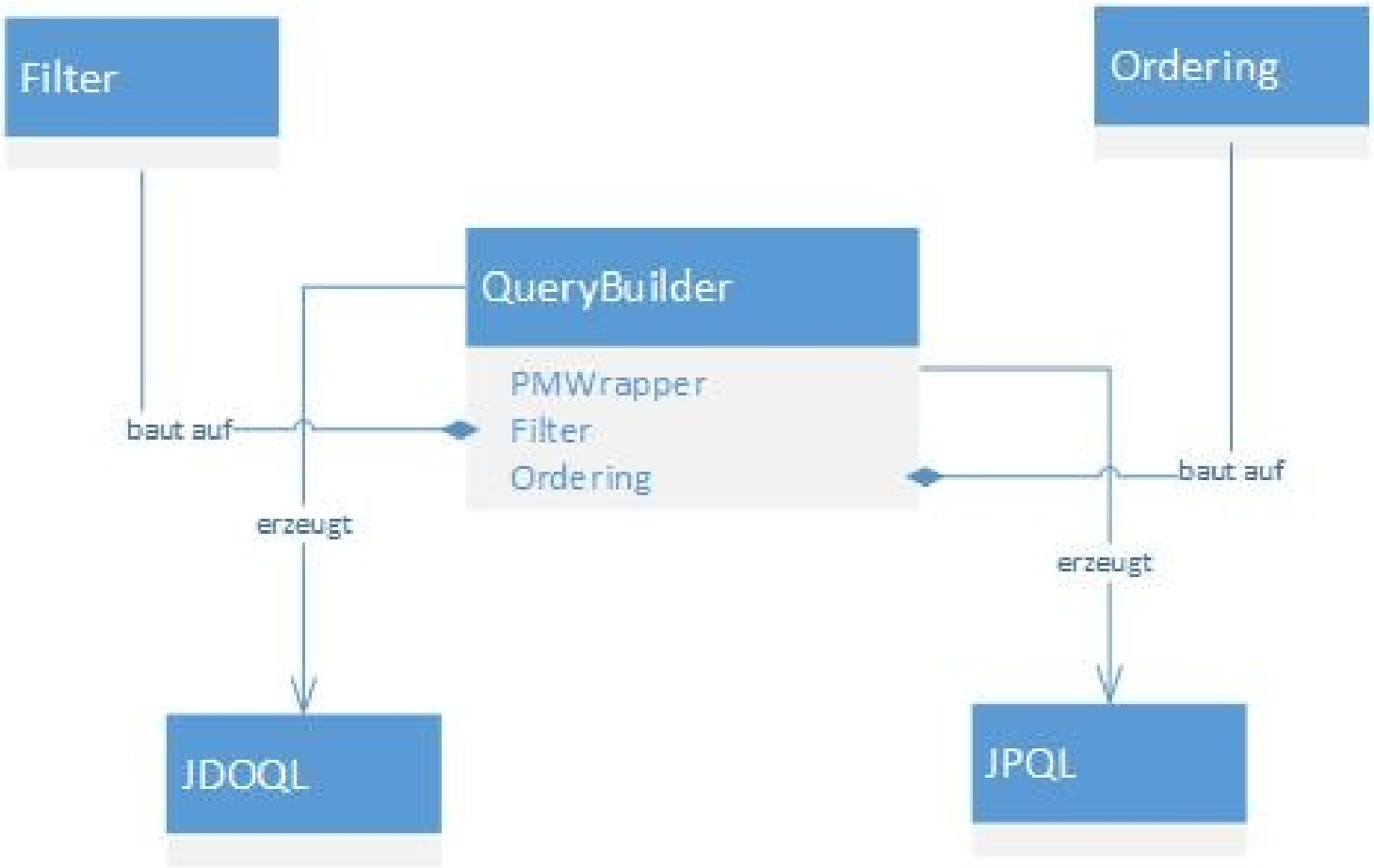
JDOQL

```
Query query = pm.newQuery (Bank.class);
query. setFilter ("customers > numberOfCustomers");
query.declareParameters(“Integer numberOfCustomers”);
List results = (List) query.execute (new Integer(1000));
```

Criteria API (JPQL)

```
CriteriaBuilder cb = em.getCriteriaBuilder();
CriteriaQuery<Object> query = cb.createQuery(Bank.class);
Root<Bank> root = query.from(Bank.class);
//Filter bauen
ParameterExpression<Integer> customers = cb.parameter(Integer.class, „customers“);
Path<List<Customer>> numberOfCustomers = root.get("customers");
Predicate condition = cb.gt(cb. size (numberOfCustomers), customers)
query. select (root);
query.where(condition);
TypedQuery<Bank> typeduery = em.createQuery(query);
typedQuery.setParameter(„customers“, Integer.valueOf(1000));
List<Bank> resultList = createQuery.getResultList();
```

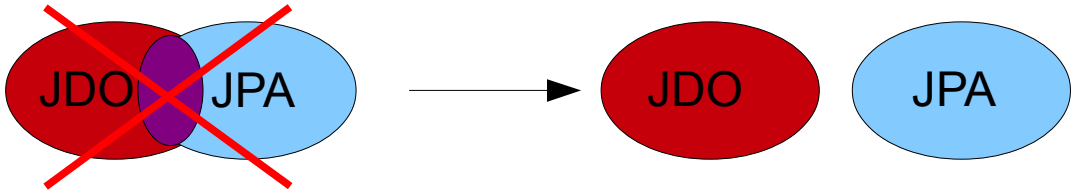
Migration - Query



Automatisierung

- Migration von Metadaten und Mapping ist zeitaufwändig
- Aber: Wiederkehrende Muster → gut automatisierbar

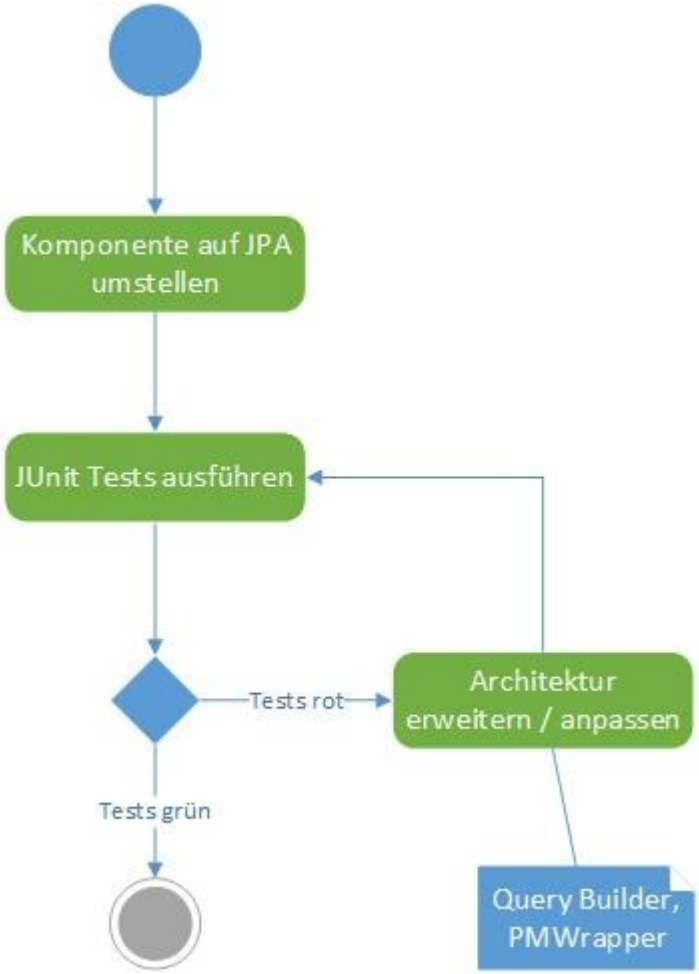
Probleme



- Vererbung zwischen JPA und JDO nicht möglich
- Attribute untereinander nicht möglich
  - Beeinflusst Migrationsstrategie
- Enhance von JDO
- Id und version von jpa

Tests

- Bestandteil der Migration
  - Wiesen auf Fehler hin
    - Architektur (PMWrapper, QueryBuilder)
    - Konfiguration (doppelte Entity, Relationen)
    - Test Driven Migration



Fazit

- Migration ist möglich
- verschiedene Strategien und Provider
- JDO und JPA schwer mischbar
- Vorhandener Tests sind sehr hilfreich

Vielen Dank für ihre Aufmerksamkeit -
Wünsche, Anregungen, Kritik ?

Quellen

Prof. Dr. Bernd Müller, Harald W.: ***JAVA PERSISTENCEAPI 2*****. Hanser Fachbuchverlag, 2012**

SolarMetric Kodo™ JDO 3.4.1 Developers Guide**. 2003**

http://jcp.org/aboutJava/communityprocess/mrel/jsr243/index3.html

http://jcp.org/en/jsr/detail?id=338