

# Generierung von J2EE-Web-Anwendungen aus visuellen Spezifikationen



Universität Paderborn

Fakultät EIM

Fachgruppe Prof. Dr. Uwe Kastens

12. Juli 2006

Andreas Koop

## Motivation

- Entwicklung von J2EE-Web-Anwendungen (neuerdings Java EE) komplex und aufwendig
- Expertenwissen aus vielen Bereichen notwendig:

| Bereich         | Benötigte Techniken                         |
|-----------------|---|
| Datenbanken     | SQL, JDBC, O/R-Mapping, ...                 |
| Web-Design      | HTML, CSS, JavaScript, ...                  |
| Web-Entwicklung | JSP, Servlets, Tiles, JSF, Spring, XML, ... |

- Wunsch: **Generierung aus visuellen Spezifikationen!**
- Entwicklung auf einem noch höheren Abstraktionsniveau möglich als mit modernen Entwurfsmustern und OO-Techniken!
- Demo

# Überblick

---

- Erweiterung des PaderWAVE-Editors
  - Typsicherheit und Eingabevalidierung
  - Datenquellen, Ausgabe dyn. Daten, Versenden von Emails
- Code-Generator für J2EE-Web-Anwendungen
  - Generierungsprozess/-ablauf
  - Architektur und eingesetzte Techniken der generierten Web-Anwendungen
- Aspekte der Realisierung
  - Datenbankoperationen mit Hilfe von DAOs und Hibernate
  - Web-Zonen/ Navigationsleiste mittels JSF-Komponenten
- Evaluierung an Beispielen: PaderPetStore, Gästebuch
  - Quantitative Analyse des generierten Codes
- Zusammenfassung und Ausblick

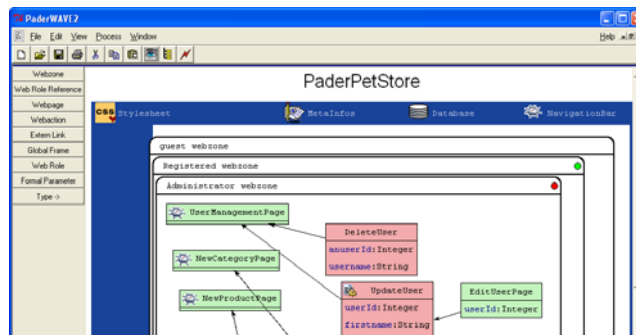
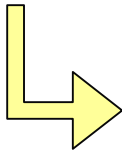
# Einordnung

## Spezifikation einer visuellen Sprache

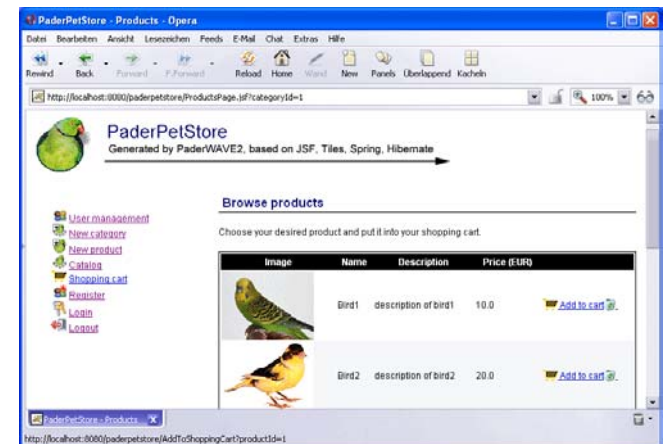
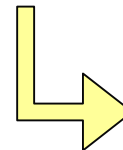
Abstrakte  
Struktur

Visuelle  
Darstellung

Analyse- und  
Transformation



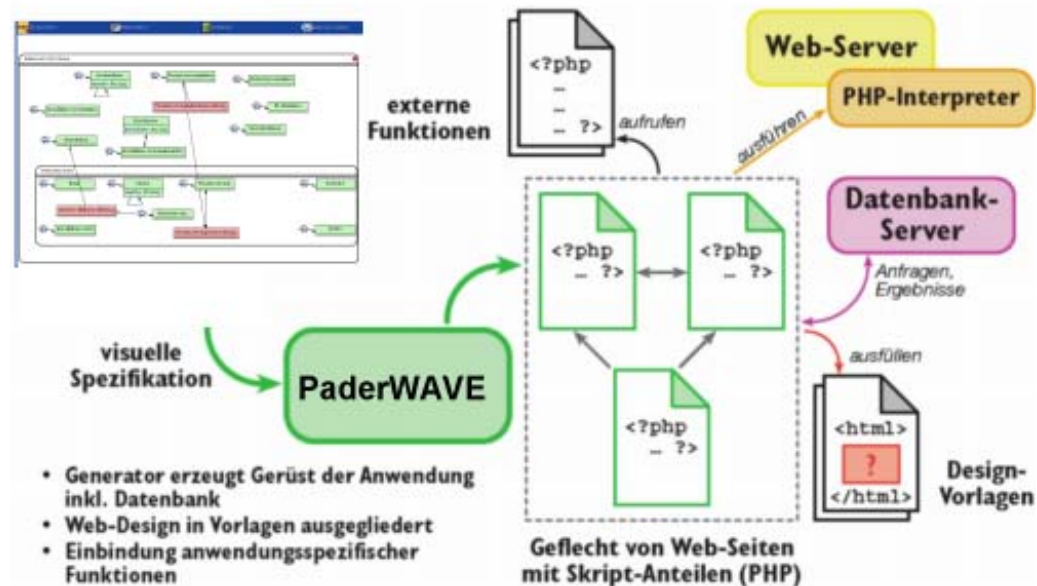
Generierung,  
Übersetzung,  
Deployment



# Erinnerung: PaderWAVE

## Paderborn Web Application builder in a Visual Environment

- CSS-Editor
- Meta-Informationen
- Datenquellen und Datenbankänderungsoperationen (nur MySQL)
- Dyn. generierte Navigationsleiste
- Web-Zonen
- Authentifizierungsmech.
- Einbettung von PHP-Code
- Wiederverwendbare Web-Seitenteile
- Parameterübergabe



<http://ag-kastens.uni-paderborn.de/lehre/paderwave/>

## PaderWAVE2 Erweiterungen im Überblick

---

- Datenbankunabhängigkeit dank Hibernate: Beispielanwendungen bislang unter HSQLDB, Firebird, MySQL getestet
- Neue Web-Operationen: Login/Logout, Emailversand, Java-Code
- Ausgereifte Eingabekonvertierung und –validierung
- Erweiterte Ausgabemöglichkeiten dynamischer Daten
  - In Tabellen- oder Listenform
  - Mit Aktionsverweisen
- Sortierbare Datenquellen
- Eingabe eines Datums mittels Kalender
- Upload und Ausgabe von Binärdaten, z.B. von Grafiken

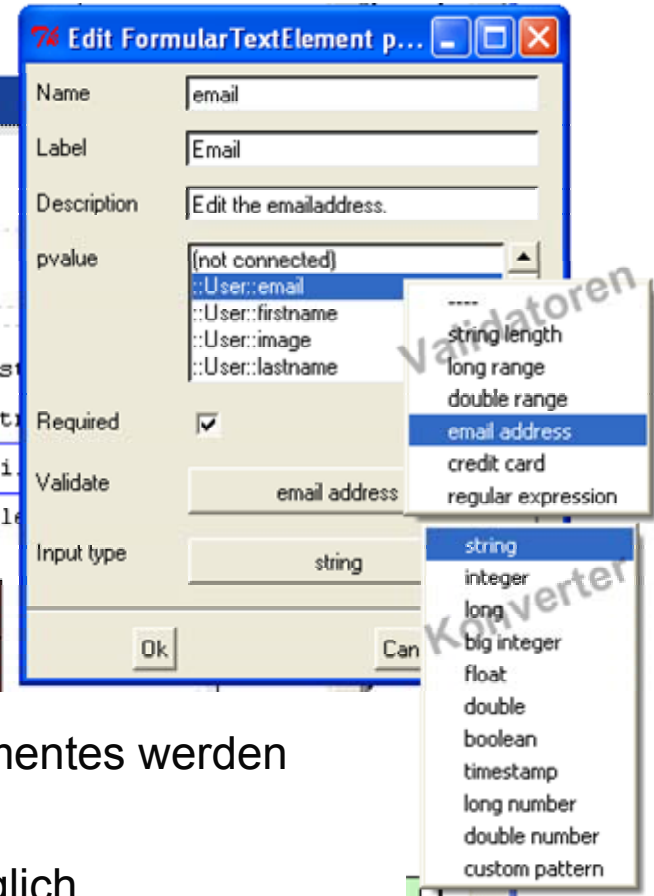
# Typsicherheit und Eingabevalidierung

- Verknüpfung von Formularelementen mit Konvertier- und Validierungsregeln möglich

Legend:  
\* information messages  
\* validation/conversion/error messages

Fields:  
Firstname\*: XYZ ✓ Edit the first  
Lastname\*: XYZ ✓ Edit the last  
Email\*: XYZ ✓ Edit the emai  
Bild: <filepath> ... Bild auswahle



Submit → `::UpdateUser`  
`::UpdateUser::userId<-userId`



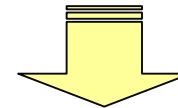
- Eingabedaten eines „typlosen“ Formularelementes werden somit typisiert
- Ausgabe von Standardfehlermeldungen möglich

# Ausgabe dynamischer Daten

- Generierung einer **Tabelle** aus der *Spezifikation einer Mustertabellenzeile*










| ::ProductsOfCategorySortedByName   |                       |                      |                       |   |
|------------------------------------|-----------------------|----------------------|-----------------------|---|
| ::Products::categoryId<-categoryId |                       |                      |                       |   |
| No.                                | Image                 | Name                 | Price (EUR)           | (no text)   |
| 1.                                 |                       |                      |                       |  ::AddToShoppingCart   |
| 2.                                 | ::Products::prodImage | ::Products::prodName | ::Products::prodPrice | Add to cart<br>::AddToShoppingCart::productId<-::Products::productId  |
| .                                  |                       |                      |                       |  ::DeleteProduct<br>(no text)<br>::DeleteProduct::productId<-::Products::productId |

- Integrierter Pagingmechanismus



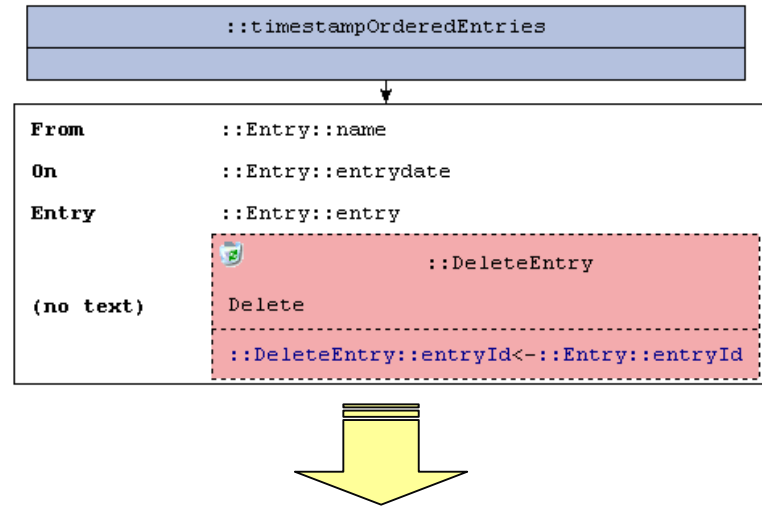
- Besondere Merkmale:
  - Nummerierungsspalte
  - Ausgabe von Binärdaten (nicht nur auf Bilder beschränkt)
  - Aktionsverweise mit Parameterübergabe

- To-Do: Sortierbare Tabelle

| No. | Image  | Name   | Price (EUR) |   |
|-----|--|--------|-------------|---|
| 1.  |    | Bird01 | 12.5        |  Add to cart      |
| 2.  |   | Bird02 | 15.5        |  Add to cart  |
| 10. |  | Bird10 | 20.0        |  Add to cart  |

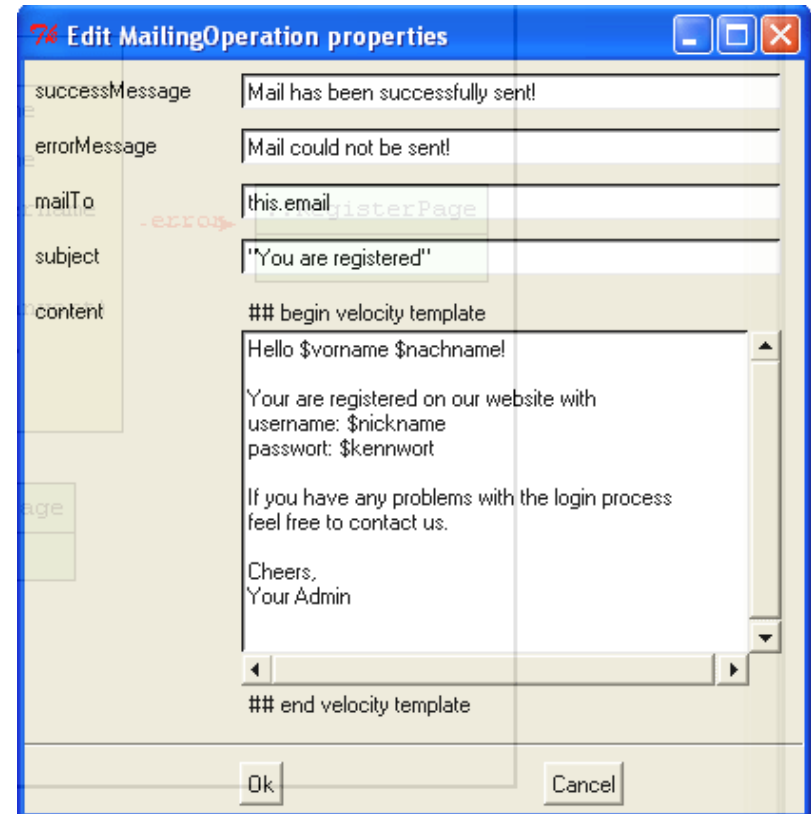
## Ausgabe dynamischer Daten (2)

- Generierung einer **Liste** aus der *Spezifikation eines Mustereintrags*
- Verwendung bei
  - Gästebüchern
  - Blogs, Kommentare
  - News
  - ....
- Transparenz: Verweise sind nur für diejenigen Benutzer sichtbar, die eine Zugriffserlaubnis auf die entsprechende Aktion besitzen



# Web-Aktion: Versenden von Emails

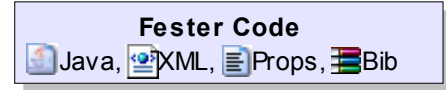
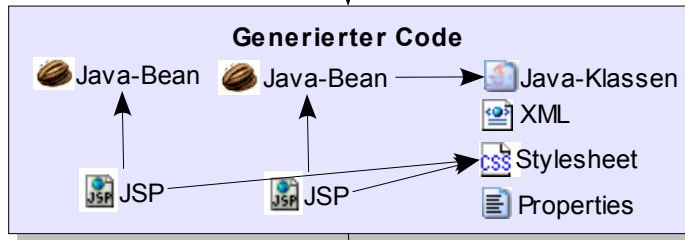
- Spezifikation von Email-Vorlagen mittels Velocity-Template
- Verwendung von Variablen für Eingabedaten aus Formularen: z.B. **\$vorname**
- Verwendung bei
  - Benutzerregistrierung
  - Bestellung
  - Bestätigung
  - ....
- Generell: Spezifikation von Erfolgs- bzw. Fehlermeldungen möglich



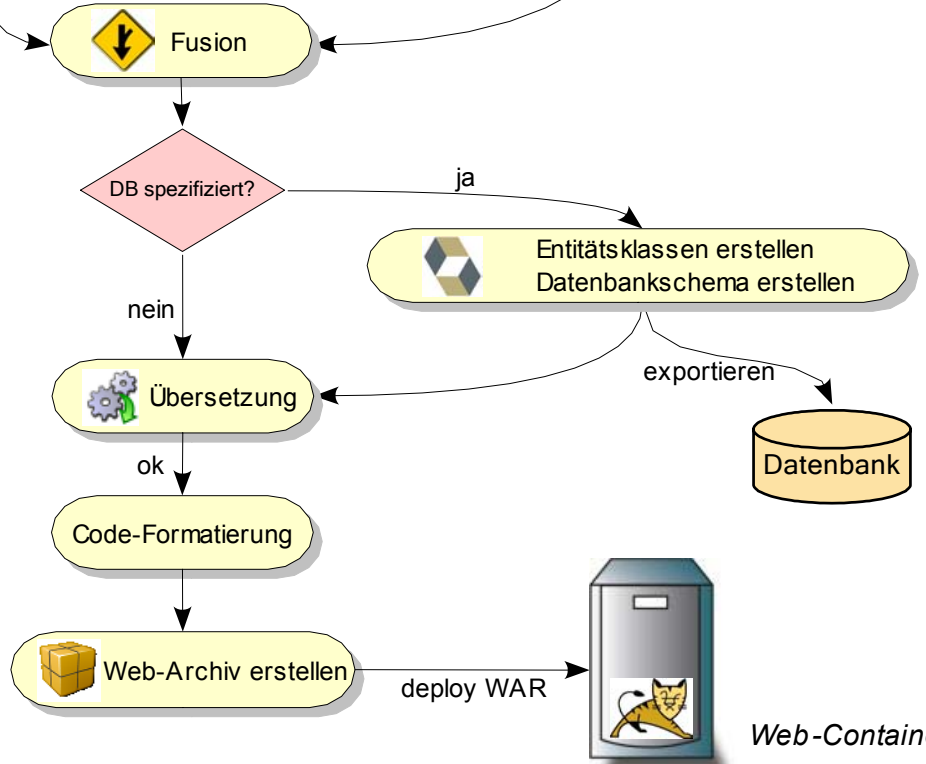


# Generierungsprozess

PTG

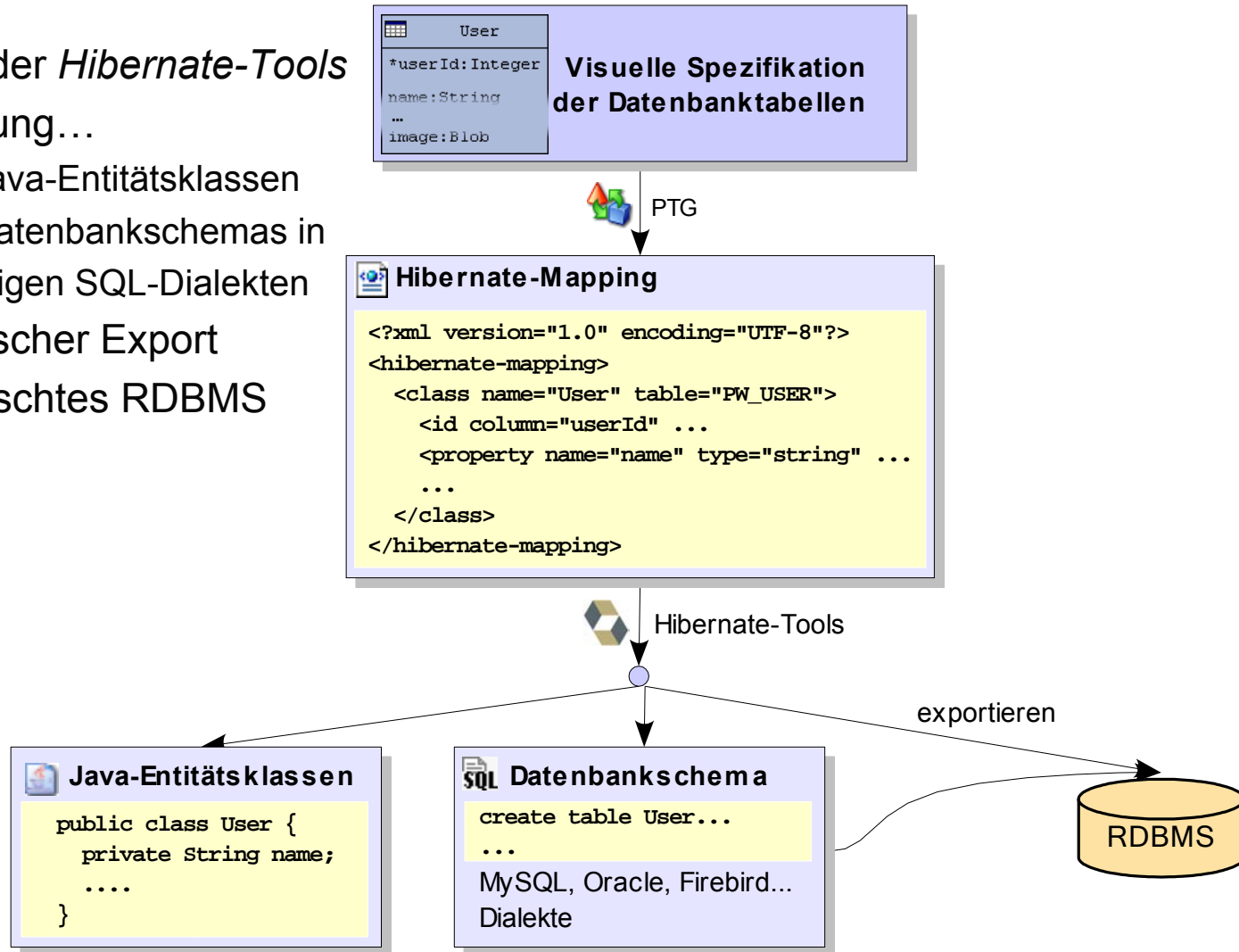


- Kontrollfluss mit Hilfe eines *Ant-Buildskripts*
- Code-Formatierung mittels *Jalopy*
- Erstellung der Entitätsklassen und des DB-Schemas mit *Hibernate-Tools*

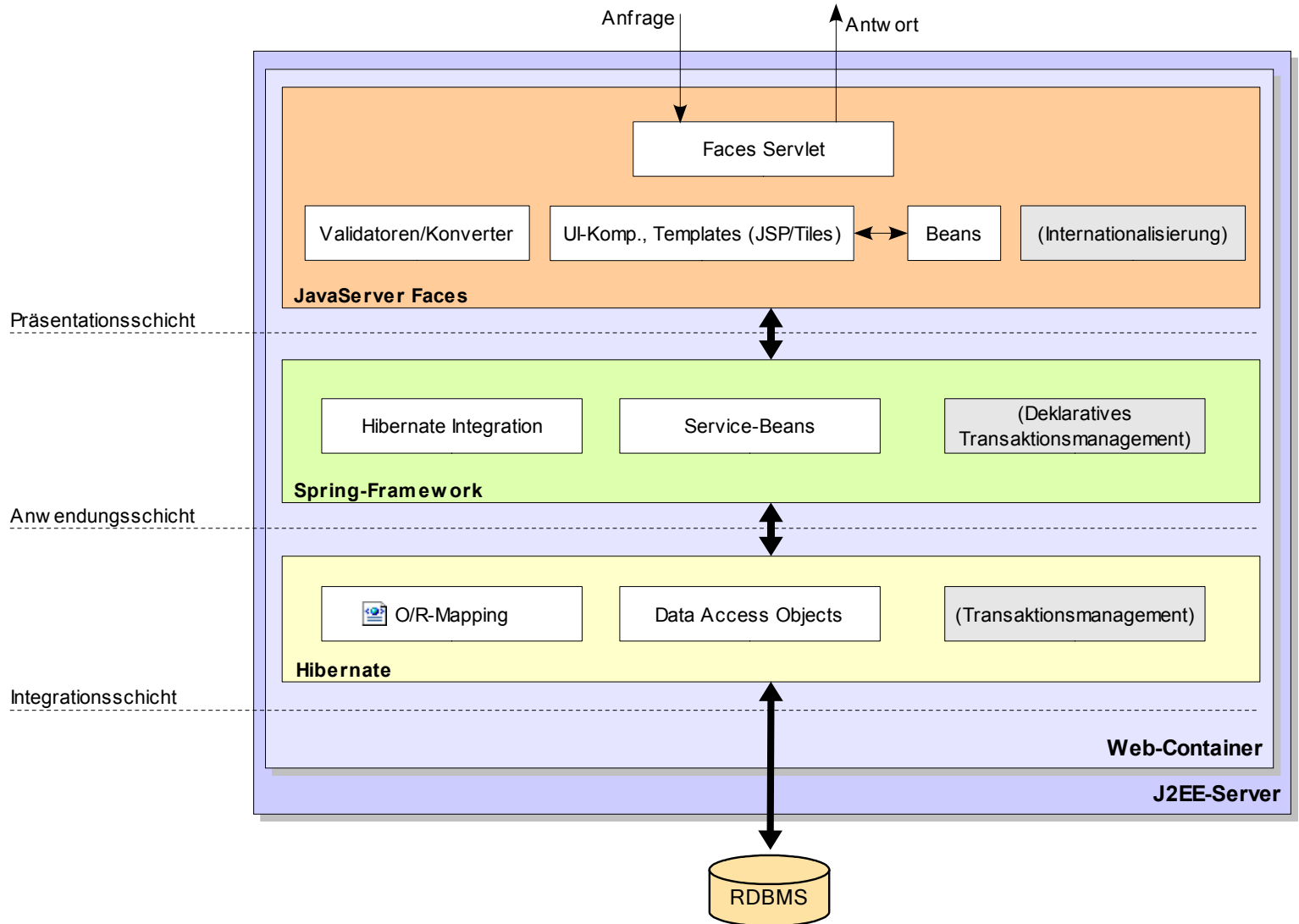


# Generierung des Datenbankschemas

- Mit Hilfe der *Hibernate-Tools* Generierung...
  - von Java-Entitätsklassen
  - des Datenbankschemas in beliebigen SQL-Dialekten
- Automatischer Export in gewünschtes RDBMS

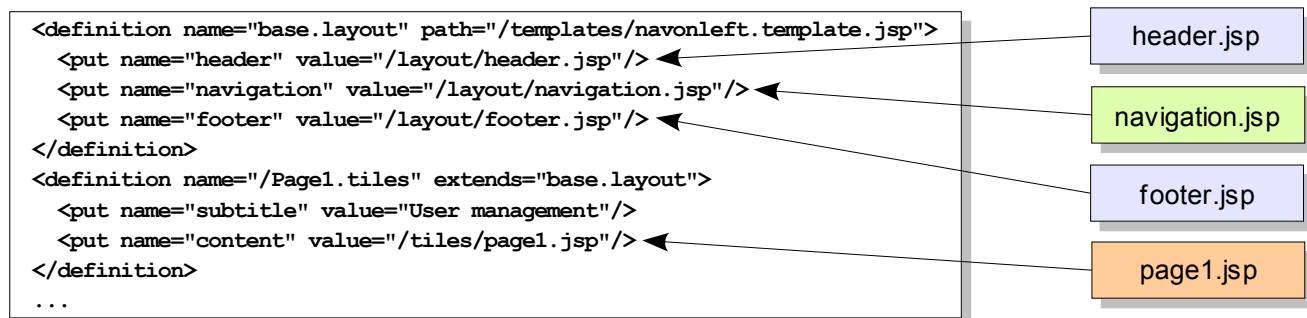
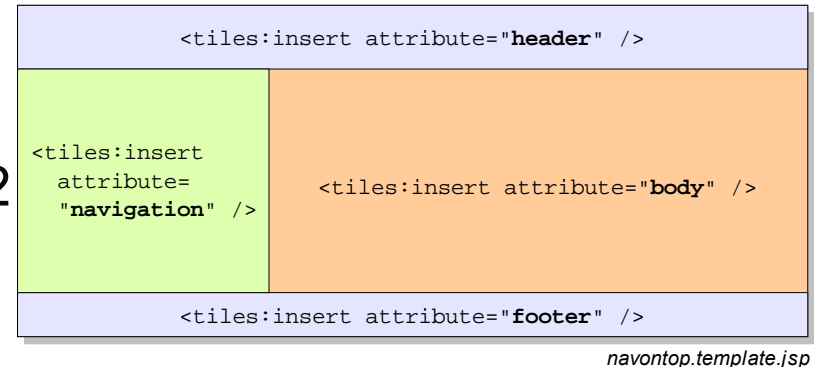


# Drei-Schichten-Architektur der generierten Web-Anwendungen



# Präsentationsschicht mit Struts Tiles/JSF

- Komposition von Webseiten mit Struts Tiles (XML-basiert)
  - 4 Basislayouts in PaderWAVE2 vordefiniert
  - Einzelne Webseitenteile sind JSPs mit JSF
- Keine Redundanzen
- Einfache Umstellung des Layouts durch Definition einer neuen Template-Seite

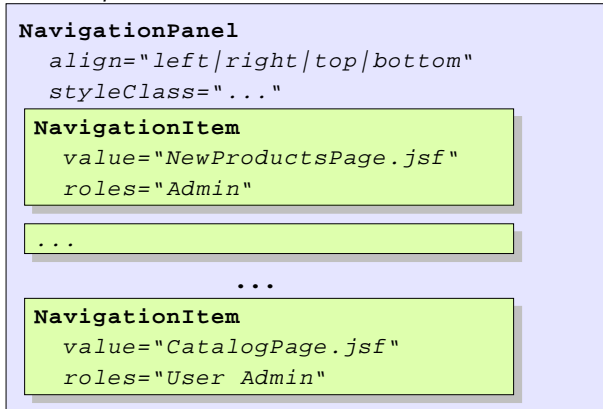


WEB-INF/tiles-def.xml

# Realisierung der dynamischen Navigationsleiste

- PaderWAVE2 generiert nur einen JSF-Komponentenbaum
  - Die eigentliche Funktionalität ist statisch im Renderer untergebracht
- Verringerte Komplexität beim Generierungsprozess

JSF-Komponentenbaum



HTML-NavigationPanelRenderer

```
public void encodeEnd(..., UIComponent comp)
    throws IOException{
    String userRoles =
        (String)sess.getAttribute(Key.USER_ROLES)
    // HTML-Ausgabe der JSF-Komponente rendern
    ...
}
```

HTML-Ausgabe (align=left)

```
<ul style="display:block;">
  <li><a href="NewProductsPage.jsf">..</a></li>
  ...
  <li><a href="CatalogPage.jsf">..</a></li>
</ul>
```

HTML-Ausgabe (align=bottom)

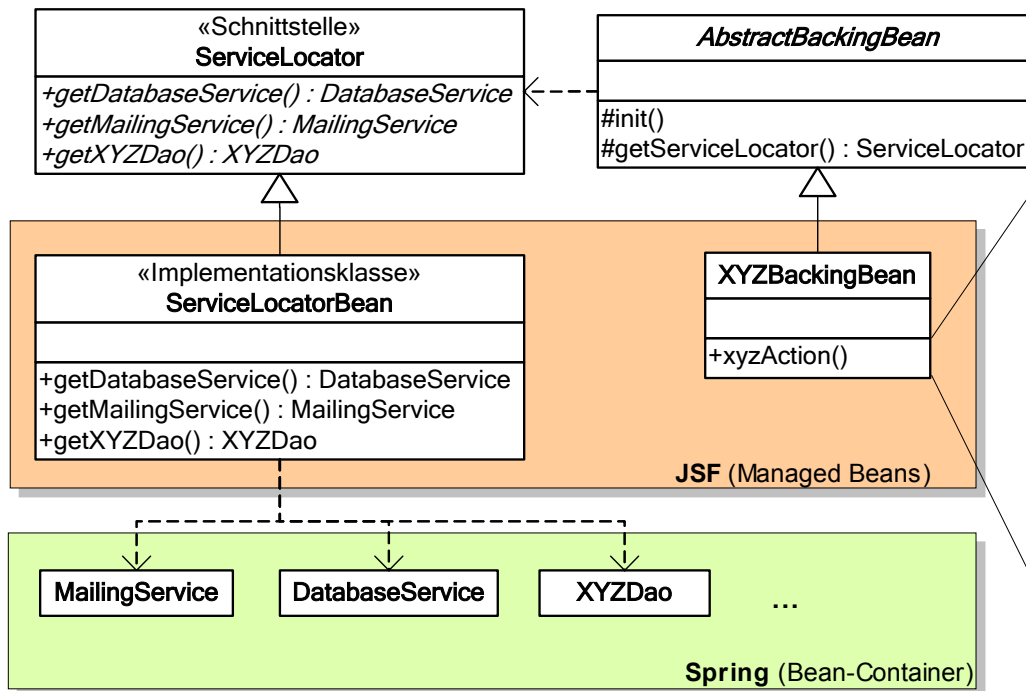
```
<ul style="display:inline;">
  <li><a href="NewProductsPage.jsf">..</a></li>
  ...
  <li><a href="CatalogPage.jsf">..</a></li>
</ul>
```

erzeugt

erzeugt

# Anwendungsschicht

- Verwendung von Spring-Beans und JSF Managed Beans zur Verwaltung von Diensten: momentan Mail- und Datenbankdienst, Data Access Objects
- Konfiguration und Initialisierung wird zentral von Spring bzw. JSF übernommen
- Einfacher und einheitlicher Zugriff: `getServiceLocator().getXYZService()...`

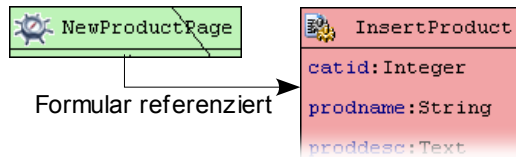


```
public String xyzAction() {
    try {
        User u = new User();
        u.setVorname(this.vorname);
        // ...und weitere Eigenschaften
        getServiceLocator().getUserDao().save(u);

        MailingJob job = new MailingJob();
        job.setSubject("Confirmation");
        job.setTo(this.email);
        job.setBody(...);
        // ...und weitere Eigenschaften
        getServiceLocator().getMailingService()
            .execute(job);
    } catch (Exception e) {
        // Ausnahme behandeln
    }
    return "LoginPage";
}
```

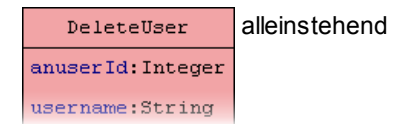
# Anwendungsschicht: Eingebettete und allein stehende Web-Aktionen

- Web-Aktionen können als Servlet oder Action-Methode einer Backing-Bean generiert werden. Für den PaderWAVE2-Benutzer völlig transparent!



wird eingebettet  
als Listener-Methode  
einer *Backing-Bean*

```
...
public class NewProductPage extends AbstractBean {
    ...
    public void insertProductAction(){
        try {
            insertProduct(this.catid, this.prodname, ..);
            ...
        } catch (Exception e) {
            ...
        }
        // forward to successpage
        return "NewProductPage";
    }
}
```

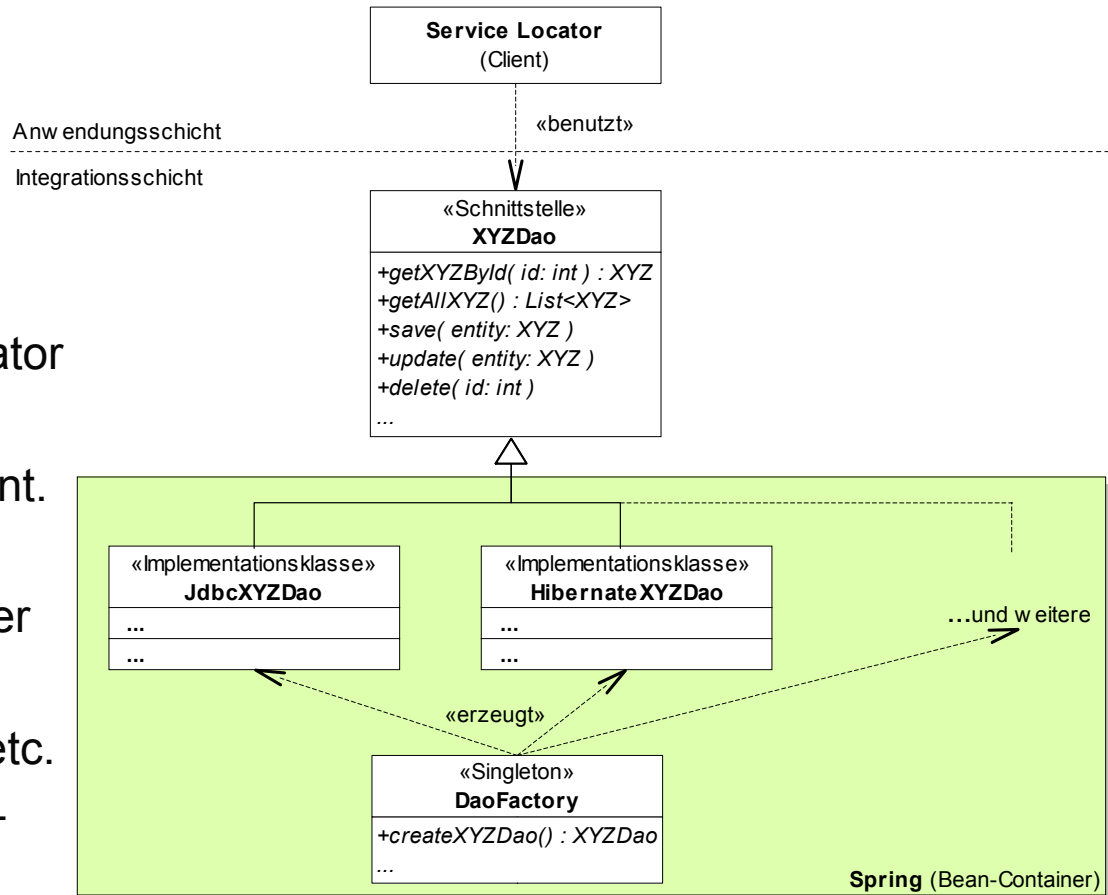


wird als  
Servlet  
realisiert

```
...
public class DeleteUserServlet
    extends ServiceLocatorServletSupport {
    ...
    protected void doGet(HttpServletRequest request, ...)
        throws ServletException, IOException {
        Integer anuserid =
            (Integer)request.getParameter("anuserid");
        try {
            deleteUser( anuserid );
            ...
        } catch (Exception e) {
            ...
        }
        // forward to successpage
        request.getRequestDispatcher("/UserMPage.jsf")
            .forward(request, response);
    }
}
```

# Integrationschicht

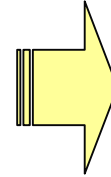
- Datenbankzugriff über Data Access Objects
  - Zu jeder spezifizierten DB-Tabelle wird ein DAO generiert
  - Der entwickelte Generator beschränkt sich bisher auf Hibernate-Implement.
- Einfacher Austausch der Datenbankschicht: z.B. JDBC, JDO, TopLink, etc.
- Datenbankunabhängigkeit ist gewährleistet!



# Realisierung von Datenbankänderungsoperationen

## ■ INSERT:

| insertUser                      |  |
|---------------------------------|--|
| Formal params                   |  |
| in_firstname:String             |  |
| in_lastname:String              |  |
| ...                             |  |
| ::User::firstname<-in_firstname |  |
| ::User::lastname<-in_lastname   |  |
| ...                             |  |



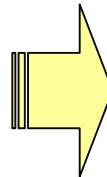
```
public void insertUser(String in_firstname,
                      String in_lastname, ...)
    throws Exception {

    User u = new User();
    u.setFirstname(in_firstname);
    u.setLastname(in_lastname);
    // ...und weitere Eigenschaften

    this.getServiceLocator().getUserDao().save(u);
}
```

## ■ UPDATE:

| updateUser              |  |
|-------------------------|--|
| Formal params           |  |
| u_userId:Integer        |  |
| u_firstname:String      |  |
| ...                     |  |
| where (userId=u_userId) |  |
| firstname<-u_firstname  |  |
| ...                     |  |
| lastname<-u_lastname    |  |



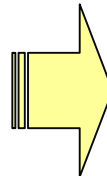
```
public void updateUser(int u_userId, String u_firstname, ...)
    throws Exception {

    Map<String, Object> _values = PWUtils.toValueMap(
        new String[]{ "u_userId", "u_firstname", ...},
        u_userId, u_firstname, ...);

    this.getHibernate().bulkUpdate(
        "update User user "
        + "set user.firstname = :u_firstname "
        + "where ( user.userid = :u_userId)", _values);
}
```

## ■ DELETE:

| X deleteUser                            |  |
|---|--|
| Formal params                           |  |
| ds_delete_userId:Integer                |  |
| ds_delete_username:String               |  |
| where (::User::userId=ds_delete_userId) |  |



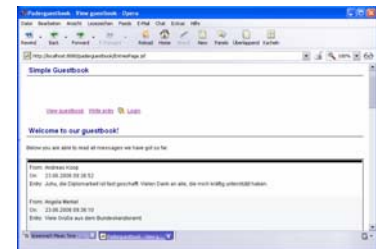
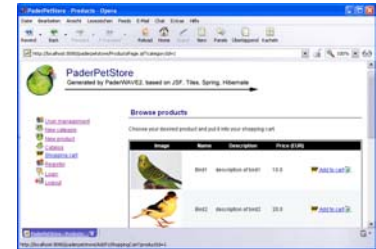
```
public void deleteUser(int del_userId, String del_username)
    throws Exception {

    Map<String, Object> _values = PWUtils.toValueMap(
        new String[]{ "del_userId", "del_username"},
        del_userId, del_username);

    this.getServiceLocator().getUserDao().delete(
        "where ( user.userid = :del_userId)", _values);
}
```

# Evaluierung an einem realistischen Beispiel

- PaderPetStore
  - Gast, Kunde, Administrator
  - Benutzerverwaltung
  - Kategorien, Produktzuordnung, persistenter Warenkorb
- Gästebuch:
  - Jeder kann Einträge hinterlassen
  - Administratoren können Einträge löschen



| Visuelles Konstrukt   | PaderPetStore | Gästebuch |
|-----------------------|---------------|-----------|
| Web-Zone              | 3             | 2         |
| Web-Seite             | 12            | 6         |
| Web-Aktion            | 12            | 7         |
| DB-Änderungsoperation | 9             | 5         |
| Datenquelle           | 11            | 4         |
| Stylesheet-Block      | 10            | 10        |

# Quantitative Analyse des generierten Codes

| Anteil                      | PaderPetStore<br>(Zeilen) | Gästebuch<br>(Zeilen) |
|-----------------------------|---------------------------|-----------------------|
| Java-Code (ohne Kommentare) | 1658                      | 996                   |
| JSP-/Velocity-Code          | 1022                      | 636                   |
| XML-/Properties-Code        | 932                       | 773                   |
| CSS-Code                    | 127                       | 127                   |
| Insgesamt                   | 3739                      | 2532                  |

- Beim PaderPetStore
  - 3,6 % überflüssige Import-Anweisungen
  - 1,3 % benutzerspezifischer Java-Code. D.h. 21 Zeilen, die jedoch fest in die visuelle Spezifikation verankert sind (Java-Code-Operation)

# Zusammenfassung

---

- Generierung kleiner bis mittelgroßer J2EE-Web-Anwendungen auf architektonisch hohem Niveau
  - Datenbankunabhängigkeit
  - Web-Operationen: Login/Logout, Emailversand, Einbettung v. Java-Code
  - Ausgereifte Eingabekonvertierung und –validierung
  - Erweiterte Ausgabemöglichkeiten dynamischer Daten
- Verwendung von Struts Tiles, JSF, Spring, Hibernate
- Ausblick
  - Transaktionen
  - Mehrsprachigkeit
  - Web-Portale aus vorgefertigten Komponenten im Baukastenprinzip generieren
  - Inkrementelle Übersetzung für die komfortable Einbettung von benutzerspezifischem Java-Code
  - Ersetzung von Struts Tiles durch Facelets

# Vielen Dank für Ihr Interesse

---

?

?

Fragen?

?

?

?

?

?

?

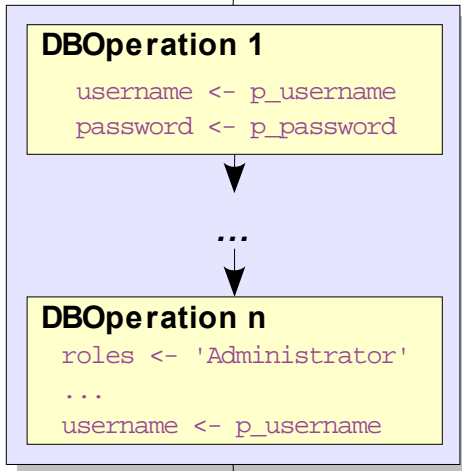
?

# Es folgen Backup-Folien

# Backup-Folie 1: Transaktionsmanagement (Ausblick)

- Deklaratives Transaktionsmanagement mittels Spring AOP

## Transaktion



## Bean

```
public void transactionalOp( ... ) {
    dbOperation1( ... );
    ...
    dbOperationN( ... );
}
```

## Spring-Konfigurationsdatei

```
<bean id="beaname" class="or...TransactionProxyFactoryBean">
    ...
    <property name="target"><ref local="...Bean"/></property>
    <property name="transactionAttributes">
        <props>
            <prop key="transactional*">PROPAGATION_REQUIRED</prop>
        </props>
    </property>
</bean>
```

# Backup-Folie 2: Inkrementelle Übersetzung (Ausblick)

PaderWAVE2 (Bearbeitung der Code-Operation)

The screenshot shows the PaderWAVE2 IDE interface. On the left is a sidebar with various operation types like DB-Operation, Code-Operation, etc. The main workspace displays a Java class named `UserBean` with attributes `username` and `password`, and methods `register()` and `submitAction()`. A context menu is open over the `register()` method, showing options like `getPassword()`, `getServiceLocator()`, `getUploadedFilesDir()`, and `getUsername()`. A red box highlights the `register()` method code. A red arrow points from the `register()` method to a red box containing an error message: `ERROR: „this.get“ is not a correct statement!`. A red arrow also points from the error message back to the `register()` method. A red arrow labeled 'generieren' points from the `register()` method to a yellow box labeled 'Java-Code'. A red arrow labeled 'einblenden' points from the 'Java-Code' box to the `register()` method. A red arrow labeled 'übersetzen' points from the 'Java-Code' box to the error message box.

**Fehlermeldung zur Spezifikationszeit**

ERROR: „this.get“ is not a correct statement!

- Einblick in den generierten (Java-)Code zur Spezifikationszeit
- Unterstützung bei der Einbettung von benutzerdefiniertem (Java-)Code durch: Vervollständigungsassistenten, Fehlererkennung, etc.