

Fachhochschule Bielefeld
Fachbereich Ingenieurwissenschaften und Mathematik

Studienarbeit

Prof. Dr.-Ing. Lutz Grünwoldt

**Untersuchung der Java-EE-6-APIs und Entwicklung
einer Java-EE-basierten Prüfplanverwaltung**

von

Aleksandr Epp

9. Mai 2013

Zusammenfassung

Diese Arbeit hat zum Ziel die modernen Java-EE-APIs zu untersuchen, um diese bei der Entwicklung einer Web-Applikation zur Verwaltung der Prüfungspläne einzusetzen. Des Weiteren sollen die wichtigsten Funktionen des bestehenden Verwaltungs-Tools nachgebildet und für spätere Erweiterung dokumentiert werden.

Inhaltsverzeichnis

I	Java Platform, Enterprise Edition (Java EE)	5
1	Einleitung	5
2	Das GlassFish Projekt	5
3	APIs	5
3.1	Java Naming and Directory Interface (JNDI)	5
3.2	Java Transaction API (JTA)	5
3.3	Java Persistence API (JPA)	6
3.3.1	EclipseLink vs. Hibernate[1, 2, 3]	6
3.3.2	Persistence Entity	6
3.3.3	Annotationen[7]	6
3.3.4	Java Persistence Query Language (JPQL)	10
3.4	JavaServer Faces (JSF)	10
3.4.1	Expression Language (EL)	11
3.4.2	JSF vs. JSP[17]	11
3.4.3	Mojarra - JSF Reference Implementierung	11
3.5	PrimeFaces[9]	11
3.5.1	UI-Komponente	12
3.5.2	Managed-Beans	16
II	Entwicklung einer Java-EE-basierten Anwendung	17
4	Prüfplanverwaltung allgemein	17
5	PHP-basierte Prüfplanverwaltung	17
6	Konzeptausarbeitung	17
6.1	Abgrenzungskriterien	17
6.2	Datenbank	18
6.3	Externe Klassen	19
7	Umsetzung	19
7.1	Entwicklungsumgebung	19
7.2	Konfiguration	19
7.2.1	Datenbankverbindung	19
7.3	Benutzeroberfläche	19
7.3.1	Prüfplankalender	20
7.3.2	Prüfplantabelle	22
7.4	Entity-Klassen	22
7.4.1	Entity-Manager	22

7.4.2	User Transaction	23
7.5	Handler-Klassen	23
7.5.1	Annotationen	24
7.5.2	Private Variablen	24
7.5.3	Initialisierung	24
7.5.4	getter-/setter-Methoden	24
7.5.5	Eine Entität erzeugen/bearbeiten und persistieren/aktualisieren	24
7.5.6	Existierende Entität löschen	26
7.6	Service-Klassen	26
7.6.1	Converter	26
7.7	Util-Klassen	27
7.7.1	JPAUtil	28
7.7.2	PruefplankalenderHandler	28
7.7.3	PruefplantabelleHandler	28
7.7.4	PrintHandler	29
7.8	Login/Logout	29
7.8.1	Login-Filter	30
III	Zusammenfassung und Ausblick	30
8	Zusammenfassung	31
9	Ausblick	32
IV	Anhang	33

Teil I

Java Platform, Enterprise Edition (Java EE)

1 Einleitung

Die Prüfplanverwaltung ist ein Thema, das an jeder Hochschule immer aktuell ist. Mit der Zeit kommen neue Anforderungen an das Verwaltungs-Tool, welches ohne großen Aufwand den neuen Anforderungen angepasst werden soll. Die Plattform, auf der die Anwendung laufen wird, soll also wartungsfreundlich, wiederverwendbar und erweiterbar sein. Als moderne Plattform für die Entwicklung einer Webapplikation bietet sich JavaServer Faces API der Java EE an.

Java EE ist die Spezifikation einer Softwarearchitektur für die transaktionsbasierte Ausführung von in Java programmierten Anwendungen und insbesondere Web-Anwendungen. Java-EE-Komponenten erfordern als Laufzeitumgebung eine spezielle Infrastruktur, einen sogenannten Java EE Application Server, auf dem die Geschäftslogik läuft.

2 Das GlassFish Projekt

GlassFish ist ein Open-Source Application-Server-Projekt für Java EE. In der Version 3 ist GlassFish die Referenzimplementierung der neuen Java EE 6-Spezifikation. Als Persistenz-Schicht setzt GlassFish standardmäßig EclipseLink ein, sowie Grizzly als Webserver-Schicht, um Webinhalte zu liefern. Grizzly basiert auf einem modifiziertem Apache Tomcat Server.

3 APIs

In Java EE sind viele verschiedene APIs aufgeführt, die unterschiedlichsten Aufgaben erledigen lassen. Auf die in diesem Projekt eingesetzten APIs wird in den folgenden Kapiteln genauer eingegangen.

3.1 Java Naming and Directory Interface (JNDI)

Java Naming and Directory Interface (JNDI) ist eine Programmierschnittstelle innerhalb der Programmiersprache Java für Namensdienste und Verzeichnisdienste. Mit Hilfe dieser Schnittstelle können Daten und Objektreferenzen anhand eines Namens abgelegt und von Nutzern der Schnittstelle abgerufen werden. Die Schnittstelle ist dabei unabhängig von der tatsächlichen Implementierung.

In diesem Projekt wird über einen JNDI Namen die Datenbankverbindung angesprochen.

3.2 Java Transaction API (JTA)

Die Java Transaction API ist eine Programmierschnittstelle, welche den Einsatz verteilter Transaktionen unter Java ermöglicht. Transaktionen sind beim Einsatz einer Datenbank sehr wichtig. Sie ermöglichen Objekte in der Datenbank anzulegen, zu aktualisieren und zu löschen. Man programmiert allerdings in der Regel nicht direkt über JTA sondern über ein Framework, welches ein kontrolliertes Transaktions-Management gewährleistet. Zum Einsatz kam EclipseLink, da es die aktuelle Referenzimplementierung der Java Persistence API ist.

3.3 Java Persistence API (JPA)

Die Java Persistence API ist eine Datenbank-Schnittstelle für Java-Anwendungen. Sie vereinfacht die Zuordnung und die Übertragung von Objekten auf Datenbankeinträge.

3.3.1 EclipseLink vs. Hibernate[1, 2, 3]

Es existieren mehrere Implementierungen der JPA. Die zwei bekanntesten sind Hibernate und EclipseLink.

Hibernate ist wie EclipseLink ein Persistenzframework. Im Gegensatz zu EclipseLink wird Hibernate in zehntausenden Java-Projekten weltweit eingesetzt und ist somit die meist benutzte JPA-Implementierung. Allerdings ist EclipseLink die Referenzimplementierung der Java Persistence API und bringt laut vielen Erfahrungsberichten und Tests Vorteile mit, wie Stabilität und Einfachheit der Anwendung. Da die Verwendung der modernen Technologien im Vordergrund stand, wurde mit dem neuen EclipseLink-Framework gearbeitet, anstatt das altbewährte, gut dokumentierte aber jedoch etwas veraltete Hibernate-Framework zu verwenden.

3.3.2 Persistence Entity

Auf die Entity Beans des EJB-3.0-APIs wurde bei der Entwicklung verzichtet, da diese Technologie von Persistence Entities der JPA überholt ist.[4]

Eine Persistence Entity ist ein Plain Old Java Object (POJO), das üblicherweise auf eine einzelne Tabellen in der relationalen Datenbank abgebildet wird. Instanzen dieser Klasse entsprechen hierbei den Zeilen der Tabelle. Die Klasse besitzt außerdem die getter- und setter-Methoden, welche es ermöglichen die einzelnen Eigenschaften einer Instanz (Spalten eines Datensatzes in der Datenbank) auszulesen oder zu ändern. Durch das Erzeugen einer Instanz dieser Klasse und die anschließende Persistierung dieser Instanz wird ein neuer Datensatz erzeugt.

3.3.3 Annotationen[7]

Als Annotation wird im Zusammenhang mit der Programmiersprache Java ein Sprachelement bezeichnet, das die Einbindung von Metadaten in dem Quelltext erlaubt.

Ein Beispiel einer Anwendung der Annotationen ist das „Mapping“. Unter Mapping versteht man das Verknüpfen von Entities mit den Datenbanktabellen. Bevor die Annotationen eingeführt wurden, geschah dies über eine XML-Konfigurationsdatei[6]:

- EclipseLink: orm.xml
- Hibernate: hibernate.cfg.xml

Die Konfigurationsdateien sehen sehr ähnlich aus und haben die Aufgabe die Spalten einer Tabelle auf die Eigenschaften einer Persistenz-Entity abzubilden, die sogenannten „Named Queries“ (vordefinierte SQL-Abfragen) zu deklarieren etc.:

Auszug aus eclipselink-orm.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <entity-mappings version="2.1"
3   xmlns="http://www.eclipse.org/eclipselink/xsds/persistence/orm"
4   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
5
6   <package>org.bielefeld.epp.entity</package>
7
8   <named-query name="Mitarbeiter.findAll">
9     <query>SELECT m FROM Mitarbeiter m</query>
10    <hint name="eclipselink.query-results-cache" value="True" />
11  </named-query>
12
13  <named-query name="Employee.findByMaID">
14    <query>SELECT m FROM Mitarbeiter m WHERE m.maID = :maID</query>
15  </named-query>
16
17  <entity class="Mitarbeiter" access="VIRTUAL">
18    <attributes>
19      <id name="maID" attribute-type="Integer">
20        <column name="maID" />
21        <generated-value strategy="IDENTITY"/>
22      </id>
23      <basic name="maName" attribute-type="String">
24        <column name="MaName" />
25      </basic>
26      <basic name="maVorname" attribute-type="String">
27        <column name="MaVorname" />
28      </basic>
29      <basic name="maTitel" attribute-type="String">
30        <column name="MaTitel" />
31      </basic>
32      <basic name="maKurz" attribute-type="String">
33        <column name="MaKurz" />
34      </basic>
35      <many-to-one name="accID" fetch="EAGER" optional="true"
36        target-entity="Accounts">
37        <join-column name="AccID" />
38      </many-to-one>
39    </attributes>
40  </entity>
41 </entity-mappings>
```

Wie man sieht, kann die Erstellung solch einer Datei ziemlich aufwendig werden, insbesondere wenn mehrere verbundene Tabellen vorliegen. Alle Tabellen und Named Queries sind in einer Datei aufgeführt, was oft zur Verwirrung und langer Suche führt.

Annotationen haben zwar die gleiche Aufgabe, wie die einzelnen Knoten einer XML-Konfigurationsdatei,

jedoch können diese direkt bei der verbundenen Variable angegeben werden und nicht in einer separaten Datei. Das verschafft dem Entwickler einen besseren Überblick und die Mappings unterschiedlicher Tabellen werden nicht zusammen gemischt, sondern befinden sich in den entsprechenden Entity-Klassen. Die wichtigsten Entity-Annotationen sind in der Tabelle aufgeführt:

Annotation	Beschreibung
@Entity	Kennzeichnet eine Klasse als eine Entity.
@Table	Definiert die verknüpfte Tabelle.
@NamedQueries	Definiert mehrere JPQL-Abfragen.
@NamedQuery	Definiert eine statischen JPQL-Abfrage. Die Abfrage kann dann in einer Java-Klasse von einem Entity-Manager über ihren Namen aufgerufen und ausgeführt werden.
@Id	Definiert den Primärschlüssel einer Entity.
@GeneratedValue	Definiert wie der Primärschlüssel generiert wird. Nur in Verbindung mit @Id.
@Basic	Verknüpft ein Attribut mit einer Tabellenspalte.
@NotNull	Attribut darf nicht null sein.
@Column	Definiert die Tabellenspalte mit der ein Attribut verknüpft wird.
@JoinColumn	Definiert ein Attribut als ein Fremdschlüssel. Nur in Verbindung mit einer Relationship-Annotation.
@ManyToOne	Definiert eine n zu 1 Beziehung.
@OneToMany	Definiert eine 1 zu n Beziehung.
@ManyToMany	Definiert eine n zu n Beziehung.
@OneToOne	Definiert eine 1 zu 1 Beziehung.

Tabelle 1: Entity Annotationen

Der unten aufgeführte Auszug aus Mitarbeiter-Entity-Klasse veranschaulicht die Verwendung von Annotationen:

Auszug aus Mitarbeiter.java

```
1 package org.bielefeld.epp.entity;
2
3 import ...
4
5 @Entity
6 @Table(name = "mitarbeiter")
7 @XmlRootElement
8 @NamedQueries({
9     @NamedQuery(name = "Mitarbeiter.findAll", query = "SELECT m FROM
Mitarbeiter m"),
10    @NamedQuery(name = "Mitarbeiter.findByMaID", query = "SELECT m FROM
Mitarbeiter m WHERE m.maID = :maID")}
11 public class Mitarbeiter implements Serializable {
12     @Id
13     @GeneratedValue(strategy = GenerationType.IDENTITY)
14     @Basic(optional = false)
15     @Column(name = "MaID")
16     private Integer maID;
17     @Basic(optional = false)
18     @NotNull
19     @Column(name = "MaName")
20     private String maName;
21     @Basic(optional = false)
22     @NotNull
23     @Column(name = "MaVorname")
24     private String maVorname;
25     @Basic(optional = true)
26     @Column(name = "MaTitel")
27     private String maTitel;
28     @Basic(optional = true)
29     @Column(name = "MaKurz")
30     private String maKurz;
31     @JoinColumn(name = "AccID", referencedColumnName = "AccID")
32     @ManyToOne(optional = true)
33     private Accounts accID;
34     /*
35     Constructors, getters and setters
36     */
37 }
```

Neben der Annotationen aus Tab. 1 gibt es viele weitere, die in unterschiedlichsten Kontexten verwendet werden. Diese werden an den entsprechenden Stellen in dieser Dokumentation erläutert.

3.3.4 Java Persistence Query Language (JPQL)

Die Java Persistence Query Language (JPQL) wird genutzt, um Abfragen bezüglich der in der Datenbank gespeicherten Entitäten durchzuführen. Diese Abfragen ähneln syntaktisch SQL-Abfragen, beziehen sich aber auf Entitäten statt auf Datenbanktabellen.

Die JPA-Implementierungen überführen die in JPQL formulierten Abfragen zur Laufzeit in ein SQL-Statement, das vom Zieldatenbanksystem verstanden wird. Durch diese Abstraktion kann das Datenbanksystem transparent ausgetauscht werden, während die Java Klassen vollständig erhalten bleiben. Im Unterschied dazu erlaubt die JPA auch die Verwendung von "normalen" SQL-Abfragen, wobei diese als Native Query bezeichnet werden. Beim Einsatz von Native Queries muss jedoch der Anwender selbst darauf achten, dass die Abfrage vom Zielsystem verstanden wird.

Um den Unterschied zwischen SQL und JPQL zu zeigen, ist unten eine Abfrage in beiden Sprachen aufgeführt:

JPQL vs. SQL

```
1  /*JPQL*/
2
3  SELECT ppe FROM Pruefplaneintrag ppe
4  JOIN ppe.pruefPeriode pp
5  JOIN ppe.sgmId sgm
6  JOIN sgm.sgid sg
7  WHERE pp.prPeID = ?
8  AND sgm.modSem = ?
9  AND sg.sgid = ?
10
11 /*SQL*/
12
13 SELECT * FROM pruefplaneintrag ppe
14 INNER JOIN pruefperiode pp ON pp.PrPeID = ppe.PruefPeriode
15 INNER JOIN sgmodul sgm ON sgm.SGMID = ppe.SGMID
16 INNER JOIN studiengang sg ON sg.SGID = ppe.SGID
17 WHERE pp.prPeID = ?
18 AND sgm.modSem = ?
19 AND sg.sgid = ?
```

3.4 JavaServer Faces (JSF)

JavaServer Faces (kurz JSF) ist ein Framework-Standard zur Entwicklung von grafischen Benutzeroberflächen für Webapplikationen. Basierend auf Servlets und JSP-Technik, gehört JSF zu den Webtechnologien der Java Platform, Enterprise Edition (Java EE). Mit Hilfe von JSF kann der Entwickler auf einfache Art und Weise Komponenten für Benutzerschnittstellen in Webseiten einbinden und die Navigation definieren. Voraussetzungen für die Entwicklung von JSF-Content sind das JDK, ein Servlet-Container (z. B. Apache Tomcat) und Grundwissen in HTML, HTTP und der Programmiersprache Java. Zur Vereinfachung der Entwicklung kann eine Integrierte Entwicklungsumgebung (IDE) verwendet werden.

3.4.1 Expression Language (EL)

Expression Language ist eine Skriptsprache die es ermöglicht Java-Klassen und deren Methoden auf JSP- und JSF-Seiten aufzurufen. Die Sprache wurde entwickelt um den Java-Code von der Web-Seite zu trennen. JSF benutzt eine eigene auf JSP-Technik basierte Expression Language.

3.4.2 JSF vs. JSP[17]

JavaServer Pages (JSP) ist eine Web-Programmiersprache zur einfachen dynamischen Erzeugung von HTML- und XML-Ausgaben eines Webservers. Auch JavaServer Faces ist ein Framework-Standard zur Entwicklung von grafischen Benutzeroberflächen für Webapplikationen. Dennoch bringt JSF viele Vorteile mit sich:

- JSP hat kein Navigationskonzept im Sinne von JSF. D.h. man müsste Struts o.ä. einsetzen um Aktionen auszuführen, Inhalte dynamisch zu laden etc. In JSF lässt sich das sehr einfach mit Managed-Beans machen.
- JSF hat sehr viele fertige GUI-Elemente, die dem Programmierer sehr viel Arbeit abnehmen. JSP alleine hat diese nicht und es müssten wieder Struts oder Spring zum Einsatz kommen.
- Die Expression Language von JSF hat eine bidirektionale Wertebindung, d. h. sie kann Properties (Eigenschaften einer Bean) lesen (z. B. Initialisierung) und schreiben (z. B. Werte an Property binden). Zum Beispiel wird der Mitarbeitername bei einem Submit in die Variable currentMaName geschrieben und beim Laden wird er daraus gelesen:

Input-Feld

```
1 <p:inputText value="#{mitarbeiterHandler.currentMaName}"/>
```

Der zweite Punkt kann jedoch auch als Nachteil betrachtet werden. GUI-Elemente sehen zwar sehr gut aus aber möchte man selber etwas verändern oder anpassen, so fühlt man sich wie in einem Korsett eingezwängt, denn viel ändern kann man da nicht. Und auch wenn etwas angepasst werden kann, so muss man lange mit Firebug nach der richtigen DIV im DOM suchen. Insbesondere wenn man mit Frameworks wie PrimeFaces oder ICEFaces arbeitet.

3.4.3 Mojarra - JSF Reference Implementierung

Die JavaServer-Faces-Spezifikation wird von einer Reihe Frameworks implementiert. Diese werden meist in JEE-Containern mitgeliefert und lassen sich oft nur unter Schwierigkeiten austauschen. Mojarra ist die Referenzimplementierung von JavaServer Faces im Rahmen des GlassFish-Java-Anwendungsservers.

3.5 PrimeFaces[9]

PrimeFaces ist ein open source Framework, welches eine Großzahl an UI-Komponenten, eine, bei den meisten Komponenten eingebaute, Ajax-Funktionalität und sogar eine Skinning-Funktion zur Verfügung stellt. Die Dokumentation von PrimeFaces ist sehr umfangreich, behandelt jedoch nicht alle Aspekte und Eigenschaften der PrimeFaces-Komponenten. Man findet aber sehr schnell Antworten in dem Forum[18] auf der Homepage.

3.5.1 UI-Komponente

- **Layout[10]**

Unterteilt die Seite in 1 bis 5 Bereiche. Jeder Bereich wird durch ein **p:layoutUnit** dargestellt, welches die Eigenschaft **position** hat. Diese Eigenschaft kann entweder die Werte: left, right, center etc. oder west, east, middle etc. annehmen. Man muss sich also nicht darum kümmern wie groß die Navigation oder der Content sein sollen wenn sich die Fenstergröße ändert. Das übernimmt **p:layout**.

Implementierung

```
1 <p:layout fullPage="true">
2   <p:layoutUnit position="north">
3     <!-- Header -->
4   </p:layoutUnit>
5
6   <p:layoutUnit position="south">
7     <!-- Footer -->
8   </p:layoutUnit>
9
10  <p:layoutUnit position="west">
11    <!-- Navigation -->
12  </p:layoutUnit>
13
14  <p:layoutUnit position="middle">
15    <!-- Content -->
16  </p:layoutUnit>
17 </p:layout>
```

- **Menu[11]**

Mit **p:menu** wurde die Navigation implementiert. Es können Menüpunkte und Sub-Menus erstellt werden.

Implementierung

```
1 <p:menu>
2   <p:menuitem value="Startseite"
3     action="overview_standard.xhtml?faces-redirect=true"/>
4   <p:menuitem value="Prufplankalender"
5     action="ppcalendar.xhtml?faces-redirect=true"/>
6   <p:menuitem value="Prufplantabelle"
7     action="pptable.xhtml?faces-redirect=true"/>
8   <p:menuitem value="Druckvorschau"
9     action="print.xhtml?faces-redirect=true"/>
10 </p:menu>
```

- **Dialog[12]**

Ein Dialog der durch JavaScript-Funktionen .show() und .hide() angezeigt und geschlossen werden kann, kommt in beinahe allen Funktionen des Tools zum Einsatz.

- **DataTable[13]**

Iteriert durch eine bei value eingegebene Liste der Objekte.

Implementierung

```
1 <p:dataTable var="mitarbeiter"
2   value="#{mitarbeiterHandler.getAllMitarbeiter()}"
3   selection="#{mitarbeiterHandler.selectedMitarbeiter}"
4   filteredValue="#{mitarbeiterHandler.filteredMitarbeiter}"
5   rowKey="#{mitarbeiter.maID}"
6   rows="25" paginator="true"
7   sortMode="multiple">
8 <p:ajax event="rowSelect"
9   listener="#{mitarbeiterHandler.editMitarbeiter()}"
10  update=":dialogPanel"
11  oncomplete="dlg_new_entity.show()"/>
12 <f:facet name="header">
13   <p:commandButton icon="ui-icon-closethick"
14     onclick="dlg_confirm_delete.show()"
15     styleClass="entity_remove"
16     value="Ausgewaehlte loeschen"
17     update=":form_confirm_delete:displayMulti"/>
18   <p:commandButton icon="ui-icon-plusthick"
19     oncomplete="dlg_new_entity.show()"
20     actionListener="#{mitarbeiterHandler.newMitarbeiter()}"
21     styleClass="entity_add"
22     value="Neuer Mitarbeiter"
23     update=":dialogPanel"/>
24 </f:facet>
25 <p:column selectionMode="multiple"/>
26 <p:column headerText="Name" sortBy="#{mitarbeiter.maName}">
27   <h:outputText value="#{mitarbeiter.maName}, #{mitarbeiter.maVorname}"
28   />
29 </p:column>
30 <p:column headerText="Titel" sortBy="#{mitarbeiter.maTitel}">
31   <h:outputText value="#{mitarbeiter.maTitel}" />
32 </p:column>
33 <p:column headerText="KÃ¶rzel" sortBy="#{mitarbeiter.maKurz}">
34   <h:outputText value="#{mitarbeiter.maKurz}" />
35 </p:column>
36 </p:dataTable>
```

- **Kalender[14]**

Ein jQuery **DatePicker**, das Objekte vom Typ **Date** erzeugt und darstellt:

Implementierung

```
1 <p:calendar id="date"
2   stepHour="1"
3   stepMinute="5"
4   minHour="8"
5   maxHour="18"
6   showButtonPanel="true"
7   pattern="dd/MM/yyyy HH:mm"
8   value="#{pruefplanKalenderHandler.currentPPEdatZeit}"
9   mindate="#{pruefplanKalenderHandler.minDate}"
10  maxdate="#{pruefplanKalenderHandler.maxDate}"
11  required="true"
12  locale="de"
13  showWeek="true"/>
```

- **Printer[15]**

Da die Umsetzung des Projektes das Drucken über die Browser-Funktion nicht zulässt (aufgrund von **p:layout**), hat diese Funktion der Printer von PrimeFaces übernommen. Bei target-Attribut des Printers muss nur die **ID** des zu druckenden Elementes eingegeben werden. Falls mehrere Elemente gedruckt werden sollen, können diese in einem unsichtbaren Element zusammengefasst werden, z.B. **p:outputPanel** und dem Printer die ID des Panels übergeben werden:

Implementierung

```
1 <p:commandButton value="Drucken"
2   icon="ui-icon-print">
3   <p:printer target="panel_print" />
4 </p:commandButton>
5
```

- **SelectOneMenu[16]**

Auf den ersten Blick ein einfaches HTML-Select mit komplizierter Deklaration. Beim genauen Betrachten von diesem Elements stellt sich heraus, dass die Einträge nicht nur Zahlen oder Strings sein können, sondern beliebige Objekte. Dafür muss eine entsprechende Converter-Klasse geschrieben werden, welche Objekte in String umwandelt und umgekehrt:

Implementierung

```
1 <p:selectOneMenu id="erstpruefer"
2     value="#{pruefplanKalenderHandler.currentErstPruefID}"
3     required="true"
4     onchange="this.blur()">
5 <f:selectItem itemValue="#{pruefplanKalenderHandler.currentSgmodul.prID}"
6     itemLabel="#{pruefplanKalenderHandler.currentSgmodul.prID.prName},
7     #{pruefplanKalenderHandler.currentSgmodul.prID.prVorname}"/>
8 <f:selectItems value="#{prueferHandler.pruefer}"
9     var="pr1"
10    itemValue="#{pr1}"
11    itemLabel="#{pr1.prName}, #{pr1.prVorname}"/>
12 </p:selectOneMenu>
```

Die **Converter** werden im Kap. 7.6 genauer erläutert.

Die meisten Elemente wie Buttons, Selects und Links haben eingebaute Ajax-Funktionalität. All diese Komponente haben ein **actionListener-Attribut** bei dem die Methode aus der Managed-Bean angegeben wird. Diese Methode führt die gewünschte Logik aus und anschließend wird die Komponente aktualisiert, das beim **update-Attribut** aufgeführt ist (können auch mehrere sein). Des Weiteren kann mit dem Attribut **process** festgelegt werden welche Daten an das Back-End gesendet werden (@all - alle, @this - nur von dem aktuellen Element (z.B. Select-Option) oder die einzelnen IDs der Komponenten mit den Daten). Komplizierte JavaScript-Funktionen für Ajax-Aufrufe können somit vermieden und wenn man beachtet, dass nahezu alle Actions in der Anwendung Ajax-Actions sind (bis auf Navigation und Login/Logout), auch sehr viel Arbeit gespart werden.

3.5.2 Managed-Beans

Der Managed-Bean Mechanismus erlaubt es POJOs mit einem bestimmten Gültigkeitsbereich (eng. Scope) und Initialwerten für so genannte Managed Properties mit den gewünschten initialen Einträgen zu versehen. Dies kann als zusätzliche Option mit Annotations direkt im Bean konfiguriert werden. Managed-Beans können mit der Hilfe von EL-Ausdrücken direkt verwendet werden.

Die Tabelle 2. erläutert unterschiedliche Gültigkeitsbereiche:

Scope	Beschreibung
@ApplicationScoped	Für die gesamte Lebensdauer der Anwendung ist nur eine für alle Benutzer gleiche Instanz dieser Managed-Bean vorhanden.
@SessionScoped	Die Managed-Bean lebt für die Dauer einer Sitzung, in der der Benutzer mit der Anwendung verbunden ist.
@ViewScoped	Die Lebensdauer der Managed-Bean ist an die Ansicht geknüpft, in der sie verwendet wird.
@RequestScoped	Die Managed-Bean lebt für die Zeitdauer einer HTTP-Anfrage.
@NoneScoped	Die Managed-Bean wird nicht gespeichert und bei jedem Aufruf neu erstellt.

Tabelle 2: Gültigkeitsbereiche (Scope)

ApplicationScoped Bean hat also die höchste Lebensdauer und RequestScoped die niedrigste. In der Prüfplanverwaltung sind alle Managed-Beans bis auf SessionManager, welches ein SessionScoped sein muss, weil dort die Anmeldeinformationen gespeichert sind, ViewScoped.

Teil II

Entwicklung einer Java-EE-basierten Anwendung

4 Prüfplanverwaltung allgemein

Die Planung einer Prüfungsphase stellt für die Verantwortlichen einen hohen Aufwand dar. Man muss alle Prüfungen unterbringen und dabei Überschneidungen vermeiden. Prüfer können nur an einem Ort zur selben Zeit sein und Räume sind durch ihre Größe begrenzt. Viele Faktoren müssen dabei für mehrere Studiengänge beachtet werden. Schließlich soll der Ablauf der Prüfungen auch für die Prüflinge möglichst optimal sein und nicht alle Prüfungen an zwei der zehn möglichen Tage liegen. Die Ausgangslage war, dass diese Planung von Hand in Excel-Tabellen vorgenommen werden musste, was trotz viel „Kopieren&Einfügen“ sehr zeitaufwendig ist. Außerdem mussten Änderungen an den jeweiligen Stellen von Hand eingetragen werden, damit war zum Beispiel die Änderung eines Erstprüfers für ein Modul mit viel Arbeit verbunden.

5 PHP-basierte Prüfplanverwaltung

Die vorhandene Prüfplanverwaltung wurde auf Basis von PHP entwickelt. Obwohl die Funktionalität vorhanden ist, fehlt es der Anwendung an Reaktionszeit: die komplette Seite wird neu geladen wenn nur eine Tabelle aktualisiert werden soll oder gar nur ein Datensatz, jedes Mal wenn Daten abgerufen werden, wird eine Datenbankabfrage abgesetzt, weil die Daten nicht in Objekten vorliegen etc. Dennoch ist die Anwendung sehr gut programmiert worden und das Datenbankkonzept bis auf zwei Kleinigkeiten gut umgesetzt.

6 Konzeptausarbeitung

6.1 Abgrenzungskriterien

Da bei der Entwicklung Techniken eingesetzt wurden, welche im Studium nicht behandelt worden sind, stand von Anfang an fest, dass nicht die komplette Anwendung in Java EE portiert werden soll, sondern nur die wichtigsten Funktionen, sodass die Anwendung testweise eingesetzt werden kann. Folgendes soll möglich sein:

- Prüfplankalender: zwei Wochen nebeneinander anzeigen und Prüfungen eintragen;
- Prüfplantabelle: alle Prüfplaneinträge in einer Tabelle anzeigen mit der Möglichkeit diese zu sortieren und zu filtern;
- Druckvorschau: zwei Wochen untereinander, schlicht formatiert und fürs Drucken angepasst;
- Datenbankverwaltung: einzelne Tabellen aus der Datenbank bearbeiten, insbesondere Accounts und Prüfperioden anlegen;

Die nächsten Punkte werden nicht implementiert:

- Übertragen

- Prüfplanchek
- Suche
- Import/Export der Datenbank

6.2 Datenbank

Die bestehende Datenbank soll übernommen und (leider) nicht verändert werden. Dabei gibt es einen Kritikpunkt an die Datenbank:

- Das Abspeichern der Prüfplaneintrags-Konflikte funktioniert nicht korrekt bzw. funktioniert nur mit hohem Aufwand korrekt. Der Grund dafür ist, dass jede Prüfung nur einen Konflikt haben kann. Die Prüfungs-ID der Prüfung, die sich mit der aktuellen überschneidet, wird in der Spalte **Status** gespeichert. Nach nur drei Schritten, kann es sein, dass das Mechanismus nicht mehr funktioniert:

1. Zwei Prüfungen A und B anlegen die sich überschneiden. Das sieht vereinfacht so aus:

ID	Status
A	B
B	A

2. Eine Prüfung C anlegen, die sich mit A und eine Prüfung D, die sich mit B überschneidet. Jetzt sieht das so aus:

ID	Status
C	A
A	C
B	D
D	B

3. Prüfungen C und D löschen. Somit hat A kein Konflikt mehr mit C und B kein Konflikt D. Aber A und B überschneiden sich nach wie vor; denn das war die Bedingung aus dem ersten Schritt. Diese Überschneidung ist aber nicht in der Tabelle vorhanden:

ID	Status
A	kein Konfl. (weil C gelöscht)
B	kein Konfl. (weil D gelöscht)

Natürlich gibt es die Möglichkeit beim Anlegen/Verändern oder Löschen **eines** Prüfplaneintrags, anschließend **alle** Einträge untereinander auf Konflikte zu prüfen. Aber dies erscheint mir sehr irrational.

Aus diesem Grund wurde eine neue Tabelle angelegt, eine Join-Tabelle mit dem Namen **conflicts** welche nur zwei Spalten hat. In der ersten steht eine Prüfungs-ID und in der zweiten die Prüfungs-ID des Konflikts. Diese Tabelle hat also immer eine gerade Anzahl an Einträgen. Jetzt kann jede Prüfung beliebig viele Konflikte haben und alle werden festgehalten.

Neue Tabelle: conflicts

```
1 CREATE TABLE 'conflicts' (  
2   'current' int(5) NOT NULL,  
3   'conflict' int(5) NOT NULL,  
4   PRIMARY KEY ('current','conflict'  
5 ),  
6 KEY 'fk_conflicts_pruefplaneintrag1' ('current'),  
7 KEY 'fk_conflicts_pruefplaneintrag2' ('conflict'),  
8 CONSTRAINT 'fk_conflicts_pruefplaneintrag1'  
9   FOREIGN KEY ('current') REFERENCES 'pruefplaneintrag' ('PPID')  
10  ON DELETE NO ACTION ON UPDATE NO ACTION,  
11 CONSTRAINT 'fk_conflicts_pruefplaneintrag2'  
12  FOREIGN KEY ('conflict') REFERENCES 'pruefplaneintrag' ('PPID')  
13  ON DELETE NO ACTION ON UPDATE NO ACTION)  
14 ENGINE=InnoDB DEFAULT CHARSET=utf8
```

6.3 Externe Klassen

- primefaces-3.5 - die neuste PrimeFaces-Bibliothek.
- bootstrap-1.0.9 - der berühmte Bootstrap-Look für PrimeFaces.

7 Umsetzung

7.1 Entwicklungsumgebung

Die Anwendung wurde in NetBeans IDE 7.2.1 entwickelt. Als Applikations-Server wurde GlassFish der Version 3.1.2.1 eingesetzt, nicht nur weil es die Referenzimplementierung ist sondern weil es mit NetBeans mitgeliefert wird und keiner Konfiguration bedarf. Die Datenbank wurde auf einer lokalen MySQL Instanz der Version 5.5.27 aufgesetzt.

7.2 Konfiguration

7.2.1 Datenbankverbindung

Das Arbeiten mit der Datenbank erfordert eine Verbindung zum MySQL Server. Dies geschieht ganz einfach über die NetBeans. Man gibt die Server-Adresse, den Port und die Zugangsdaten und drückt auf „Connect“. Anschließend wird eine Persistenz-Einheit erstellt. Jetzt kann auf die Datenbank zugegriffen werden.

7.3 Benutzeroberfläche

Es soll nicht die komplette Benutzeroberfläche beschrieben werden, denn das Notwendige wird auch in den entsprechenden Klassen erwähnt und erläutert. Die Prüfplankalender und -tabellen werden aber trotzdem gesondert betrachtet, da es die wichtigsten Funktionen dieses Projekts sind.

7.3.1 Prüfplankalender

Auf der Web-Oberfläche ist der Prüfplankalender ein **p:outputPanel** mit zwei Tabellen.

WiSe 2011 T1

1. Semester

Informationstechnik

Aktualisieren

WiSe 2011 T1

1. Semester

Informationstechnik

Aktualisieren

1. Prüfungswoche						2. Prüfungswoche					
Uhrzeit	Mo ...	Di ...	Mi ...	Do ...	Fr ...	Uhrzeit	Mo ...	Di ...	Mi ...	Do ...	Fr ...
8:00						8:00					
9:00						9:00					
10:00						10:00					
11:00						11:00					
12:00						12:00					
13:00						13:00					
14:00						14:00					

Abbildung 1: Prüfplankalender

Jede Tabelle kann Prüfplaneinträge einer Woche anzeigen. Die Anzeige wird durch Prüfperioden-, Studiengang- und Semesterauswahl gesteuert. Hierbei zeigt die linke Tabelle nur Prüfungen der ersten Woche und die rechte nur die Prüfungen der zweiten Woche an. Es kann also nicht wie bei der PHP-Implementierung in jeder Tabelle jede beliebige Woche angezeigt werden. Durch das Drücken des „Aktualisieren“-Buttons, wird eine Ajax-Anfrage ausgelöst, und die Handler-Klasse liefert die aussortierten Prüfplaneinträge zurück.

Eine DataTable muss über eine Liste iterieren. Die Liste der aussortierten Prüfplaneinträge zu iterieren wäre aber nicht sinnvoll, da nicht alle Zellen der Tabelle gefüllt werden müssen. Deswegen geht man an dieser Stelle eine Liste mit Zeiten (8 bis 19 Uhr) durch, und vergleicht die Prüfplaneinträge mit der Zeit an der aktuellen Position. Wenn Zeit und Tag übereinstimmen, wird die Zelle gefüllt.

Durch Doppelklick in einer Zelle wird ein Dialog-Fenster geöffnet in dem man entweder eine neue Prüfung eingetragen werden kann, falls die angeklickte Zelle leer war, oder eine existierende Prüfung aus dieser Zelle bearbeiten. Falls eine Prüfung, die gerade bearbeitet wird, einen oder mehrere Konflikte hat, besteht die Möglichkeit diese aus einer Select-Liste auszuwählen. Bei der Auswahl einer Prüfung aus der Liste, gelangt man zu dieser Prüfung und kann ihre Eigenschaften bearbeiten.

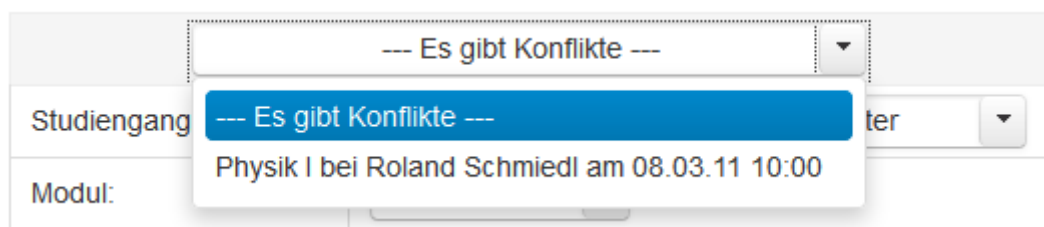


Abbildung 2: Überschneidungen

Die Eingabe-Maske hat Select-Listen mit Studiengängen, Semestern und Modulen. Dies soll für bessere Übersicht sorgen. Werte für Semester und Studiengänge werden mit den gleichnamigen Attributen des Kalenders initiiert. Anhand von diesen Werten, wird die Liste der Modulen gebildet. So muss man nicht in einer langen Liste alle im System vorhandenen Module suchen, sondern hat immer eine überschaubare Anzahl an Modulen zur Auswahl. Natürlich wird die Liste länger, wenn der Semester-Wert auf „Alle“ gesetzt ist.

Studiengang/Semester:	Informationstechnik	4. Semester
Modul:	Bitte wählen...	
Prüfperiode: *	Bitte wählen...	
Prüfungstermin: *	Automatisierungstechnik	
Erstprüfer: *	Regelungstechnik	
Zweitprüfer:	Angewandte Mathematik I	
Raum:	Angewandte Mathematik II	
	Messtechnik	

Abbildung 3: Modulauswahl

Die Terminauswahl ist auf die Tage der gewählten Prüfungsperioden begrenzt. So sieht der Anwender deutlich, an welchen 5 Tagen die Prüfung stattfinden kann.

März 2011							
Woche	Mo	Di	Mi	Do	Fr	Sa	So
9		1	2	3	4	5	6
10	7	8	9	10	11	12	13
11	14	15	16	17	18	19	20
12	21	22	23	24	25	26	27
13	28	29	30	31			

Zeit 09:00

Stunde

Minute

Schließen

Abbildung 4: Prüfperiode im Kalender

Wenn die Prüfungseigenschaften festgelegt sind, klickt man auf „Speichern“. Im Hintergrund wird geprüft ob alle Pflichtfelder gefüllt sind, sodann werden alle Eingaben an die Managed-Bean übergeben. Hier werden die Konflikte ermittelt. So sollten Konflikte bestehen so wird der Benutzer aufgefordert zu bestätigen, ob die Prüfung trotzdem angelegt werden soll oder nicht.

Prüfungen überschneiden sich ✕

⚠ Unter Vorbehalt eintragen?

Konflikt mit:

- Betriebssysteme I bei Schwenzfeier-Hellkamp am 08.03.11 11:00
- Mathematik I bei Ohlhoff am 08.03.11 9:00

Abbildung 5: Eintrag trotz Überschneidungen

Wenn keine neue Prüfung angelegt, sondern eine existierende bearbeitet worden ist, wird ebenfalls geprüft ob bestehende Konflikte aufgelöst werden konnten.

7.3.2 Prüfplantabelle

Die Prüfplantabelle funktioniert ähnlich wie der Kalender und beide Funktionen nutzen viele gleiche Methoden. Eine interessante und hoffentlich hilfreiche Eigenschaft der Prüfplantabelle ist die Möglichkeit diese nach Prüfperioden und Studiengängen zu filtern.

In der Prüfplantabelle können ebenso Prüfungen eingetragen und bestehende bearbeitet werden. Überschneidungen werden hier mit gelb markiert.

7.4 Entity-Klassen

Entity-Klassen, wie schon erwähnt, können automatisch erstellt werden. Dabei sollte man darauf achten, dass bei den verbundenen Tabellen, die Fremdschlüssel wirklich Constraints sind und nicht nur FK in den Spaltennamen haben, sonst wäre das Feld **accId** in der Klasse Mitarbeiter vom Typ Integer und nicht vom Typ Accounts. Wenn dies passt, erstellt man einfach „Entity class from Database“. Diese erhalten einen Konstruktor und getter- und setter-Methoden für die einzelnen Felder.

7.4.1 Entity-Manager

Um mit Entitäten zu arbeiten, diese zu persistieren, zu aktualisieren oder zu löschen, wird ein **Entity-Manager**(EM) benötigt. Dieser kann aus einer **EntityManagerFactory**(EMF) geholt werden. EM ist ein sehr einfaches und leichtes Objekt von dem mehrere Instanzen erzeugt werden können ohne das System zu verlangsamen. EMF ist dagegen ein sehr großes und kompliziertes Objekt in dem die Datenbankverbindung gespeichert ist, deswegen sollte dieses

Objekt nur ein Mal erzeugt werden. Dies geschieht in der Klasse **JPAUtil** und wird im Kap. 7.6 erläutert. EM hat vier wichtigen Methoden:

- **persist (Entity-Objekt)**
Persistiert eine Instanz einer Entity-Klasse. Wird bei Neuanlage eines Datenbankeintrages verwendet.
- **merge (Entity-Objekt)**
Aktualisiert (engl. merge = zusammenfügen) eine existierende Instanz einer Entity-Klasse. Wird zum Ändern eines Datenbandedintrages verwendet.
- **remove (Entity-Objekt)**
Löscht eine existierende Instanz einer Entity-Klasse.
- **find (Entity-Klasse, Entity-Objekt-ID)**
Liefert die entsprechende Entität zurück nach Übergabe der Klasse und der ID der gesuchten Entität. Bevor ein Objekt aktualisiert/geändert werden kann, muss dieser erst mit der find()-Methode aus der Datenbank geholt werden.

7.4.2 User Transaction

Mit dem Entity-Manager können also Datenbankeinträge verwaltet werden. Nun werden die Änderungen aber nicht wirksam ohne eine Transaktion auszuführen, die diese Änderungen in die Datenbank schreibt. Dafür gibt es mindestens zwei Implementierungen in Java. In diesem Projekt wurden die UserTransaction-Klasse für die Transaktionen verwendet. Vor dem Einsatz eines EMs muss eine Transaktion gestartet werden und nach dem die gewünschten Änderungen angewendet worden sind, muss diese beendet werden. Dafür gibt es die Methoden:

- **beginn()**
Startet eine Transaktion.
- **commit()**
Beendet eine Transaktion und wendet die Änderungen an der Datenbank an.

7.5 Handler-Klassen

Die Bezeichnung kommt aus dem Buch vom Herrn Bernd Müller „JavaServer Faces 2.0 Ein Arbeitsbuch für die Praxis“. Für jede Entity-Klasse gibt es eine Handler-Klasse oder Managed-Bean, in der die Objekte der Entity-Klasse persistiert, aktualisiert oder gelöscht werden können. Unter anderem besitzen diese Klasse Methoden um einzelne Eigenschaften der Objekte formatiert auszugeben. Dieses wird in den Verwaltungsfunktionen der Anwendung benutzt um z.B. in den Tabellen formatierte Ausgaben zu machen. Am Beispiel der PrueferHandler-Klasse werden die wesentlichen Aspekte erläutert.

7.5.1 Annotationen

Noch vor der Klassendeklaration stehen zwei Annotationen:

PrueferHandler: Annotationen

```
1 import ...
2
3 @ManagedBean
4 @ViewScoped
5 public class PrueferHandler implements Serializable ...
```

Nur wenn eine Klasse als Managed-Bean bezeichnet ist, kann auf diese von einer JSF Seite zugegriffen werden. ViewScoped bedeutet, dass die Bean solange Informationen speichert bis die Seite verlassen/komplett neu geladen wird.

7.5.2 Private Variablen

Wie in Java üblich, sind alle Variablen vom Zugriff von Außen geschützt und verfügen deswegen über getter- und setter-Methoden. Die Variablen mit dem Präfix **current** speichern die Eigenschaften des Objekts das gerade angelegt wird oder das Objekt selbst, falls es schon existiert.

7.5.3 Initialisierung

PrimeFaces haben einen eingebauten Mechanismus, der es ermöglicht die Inhalte einer **p:dataTable** zu filtern und zu sortieren. Allerdings sollen die Datensätze dafür, in einer Liste zwischengespeichert werden. Deswegen gibt es eine **init()**-Methode, in der alle Objekte aus der Datenbank geholt werden und in die Liste mit dem Präfix **all** eingehen (hier: List<Pruefer> allPruefer). Die Methode wird nach jeder Änderung der Datenbank aufgerufen, d.h. wenn neue Datensätze dazu kommen und alte aktualisiert oder gelöscht werden.

7.5.4 getter-/setter-Methoden

NetBeans und andere modernen IDEs haben mehrere Funktionen, die Source-Code automatisch generieren können. Diese gibt es auch für getter- und setter-Methoden. Man braucht also nur die nötigen Variablen zu deklarieren und mit ein Paar Maus-Klicks sind die genannten Methoden da.

7.5.5 Eine Entität erzeugen/bearbeiten und persistieren/aktualisieren

Um einen neuen Prüfer anzulegen drückt man zunächst auf den Button **Neuer Dozent**. Es wird als erstes eine Methode mit dem Präfix **new** (hier: newPruefer()) aufgerufen, die die current-Variablen zurücksetzt. Es öffnet sich eine Eingabe-Maske, in der die Eigenschaften eines Prüfer-Objekts bearbeitet werden können. Dabei ist zu beachten, dass es zwei mit * gekennzeichneten Pflichtfelder gibt, die auf jeden Fall gefüllt sein müssen. Die zweite Möglichkeit in diese Maske zu gelangen ist einen Dozenten in der Tabelle anzuklicken. Hierbei wird als erstes eine Methode mit dem Präfix **edit** (hier: newPruefer()) aufgerufen, die die current-Variablen

mit den Eigenschaften des gewählten Prüfers füllt. Mit dem Klick auf **Speichern** beginnt die Persistierung.

Die Methode mit dem Präfix **insertOrUpdate** erzeugt eine Instanz der Pruefer-Klasse und persistiert oder aktualisiert diese(hier: insertOrUpdatePruefer()). Nach dem ein Prüfer-Objekt erzeugt wurde, wird geprüft ob der Prüfer, dessen Eigenschaften gerade bearbeitet worden sind, bereits existiert oder neu angelegt werden soll. In beiden Fällen werden die **current**-Variablen durch die setter-Methoden in das Prüfer-Objekt geschrieben. Der Unterschied liegt darin, dass im ersten Fall der Datensatz neu angelegt werden soll, also die Entität persistiert wird. Im zweiten Fall soll ein existierender Datenbankeintrag aktualisiert werden. Dazu soll die richtige Prüfer-Instanz erst gefunden, die Eigenschaften gesetzt und anschließend die Instanz aktualisiert werden. Folgender Code zeigt die angesprochenen Programmausschnitte:

PrueferHandler: insertOrUpdatePruefer()

```
1 public void insertOrUpdatePruefer(){
2   logger.log(Level.INFO, "Start insertOrUpdatePruefer");
3
4   Pruefer p;
5   try {
6     utx.begin();
7     if(currentPruefer == null){
8       p = new Pruefer();
9       p.setPrName(currentPrName);
10      p.setPrVorname(currentPrVorname);
11      p.setPrKurz(currentPrKurz);
12      p.setPrTitel(currentPrTitel);
13      em.persist(p);
14    } else {
15      p = em.find(Pruefer.class, currentPruefer.getPrID());
16      p.setPrName(currentPrName);
17      p.setPrVorname(currentPrVorname);
18      p.setPrKurz(currentPrKurz);
19      p.setPrTitel(currentPrTitel);
20      em.merge(p);
21    }
22    utx.commit();
23
24    this.init();
25    logger.log(Level.INFO, "Success at insertOrUpdatePruefer");
26
27  } catch (Exception ex) {
28    logger.log(Level.SEVERE, null, ex);
29  }
30 }
31
```

Anschließend wird die init()-Methode aufgerufen, um die Änderungen in der Tabelle für den Anwender sichtbar zu machen.

7.5.6 Existierende Entität löschen

Mit dem Klick auf den Button **Ausgewählte löschen** wird das Löschen der ausgewählten Einträge initiiert. Dabei sind die ausgewählten Dozenten durch PrimeFaces in der Liste **selected-Pruefer** hinterlegt worden. Danach wird der Benutzer aufgefordert diese Aktion zu bestätigen und bekommt alle ausgewählten Einträge angezeigt.

Die Methode mit dem Präfix **delete** (hier: `deletePruefer()`) löscht alle Objekte aus `selected-Pruefer`, falls keine Konflikte auftreten. Ein Objekt kann nicht gelöscht werden, wenn es von einem anderen Objekt referenziert ist, d.h. dass ein Prüfer nicht gelöscht werden kann, wenn der Prüfer bei einer Prüfung oder einem Modul eingetragen ist. Dank `UserTransaction` muss man sich nicht um das `Rollback()` kümmern, denn dies geschieht automatisch, wenn ein Fehler passiert. Folgender Code zeigt die angesprochenen Programmausschnitte:

PrueferHandler: `deletePruefer()`

```
1 public void deletePruefer() {
2     try{
3         utx.begin();
4         for(Pruefer pruefer : selectedPruefer) {
5             em.remove(em.merge(pruefer));
6         }
7         utx.commit();
8     }
9     this.init();
10 } catch (Exception ex) {
11     logger.log(Level.SEVERE, null, ex);
12 }
13 }
```

Anschließend wird die `init()`-Methode aufgerufen, um die Änderungen in der Tabelle für den Anwender sichtbar zu machen.

7.6 Service-Klassen

Die Service-Klassen sind zu 100% identisch und sind in diesem Projekt nur **Converter**-Klassen.

7.6.1 Converter

Die Konvertierung ist ein wichtiger Aspekt im JSF-Lebenszyklus, da Werte am Webclient in Form von Zeichenketten, serverseitig jedoch als Java-Typen dargestellt werden. Den Konvertieren fällt dabei die Rolle eines internen Vermittlers zu. Sie kümmern sich um die Umwandlung der vom Benutzer eingegebenen Zeichenketten in Java-Objekte und wandeln Java-Objekte beim Rendern der Ausgabe wieder in Zeichenketten um. In JSF gibt es für sehr viele Datentypen bereits Standardkonverter, die automatisch zum Einsatz kommen. Wenn man zum Beispiel im Prüfplankalender das Prüfungsdatum auswählt, so sieht der Benutzer die String-Repräsentation eines **Date**-Objekts. JSF verwendet, falls kein anderer Konverter definiert ist, automatisch den Standardkonverter für den Datentyp `Date`. Dieser Standardkonverter ist für beide Richtungen der Konvertierung verantwortlich. Zum einen konvertiert er beim Rendern der Ansicht das Datum in eine Zeichenkette. Beim Absenden des Formulars konvertiert er außerdem die vom Benutzer eingegebene Zeichenkette wieder in eine `Date`-Instanz.[8]

Es ist auch möglich eigene Converter zu implementieren, die Entity-Instanzen in Strings umwandeln und umgekehrt. Da die String-Repräsentation eindeutig sein muss, eignet sich der Primärschlüssel für diese Aufgabe sehr gut. Der Code-Ausschnitt zeigt die PrueferConverter-Klasse:

```
PrueferConverter.java

1 imports ...
2
3 @FacesConverter(forClass=org.bielefeld.epp.entity.Pruefer.class)
4 public class PrueferConverter implements Converter{
5
6     @Override
7     public Object getAsObject(FacesContext context,
8     UIComponent component, String value) {
9         if (value.trim().equals("")) {
10             return null;
11         } else {
12             try {
13                 PrueferHandler prh = new PrueferHandler(JPAUtil.getEM());
14                 return prh.getPrueferByPrID(Integer.valueOf(value));
15             } catch (Exception ex) {
16                 logger.log(Level.SEVERE, null, ex);
17             }
18             return null;
19         }
20     }
21
22     @Override
23     public String getAsString(FacesContext context,
24     UIComponent component, Object value) {
25         if (value == null || value.equals("")) {
26             return "";
27         } else {
28             return String.valueOf(((Pruefer) value).getPrID());
29         }
30     }
31 }
32 }
```

Die erste Methode gibt nach Übergabe einer ID das Objekt zurück. Die zweite Methode wandelt ein Objekt in ein String um. Dabei wird die ID des Objekts als ein String zurückgegeben. Converter werden von allen Eingabe-Elementen benutzt: Textfelder, Select-Listen etc.

7.7 Util-Klassen

Die Klassen aus dem **util**-Package erfüllen die wichtigsten und kompliziertesten Aufgaben dieser Anwendung. Alle Util-Klassen bis auf JPAUtil, sind Managed-Beans, da sie die Informationen für Webclients bereitstellen.

7.7.1 JPAUtil

JPAUtil ist die kleinste Klasse. Sie hat eine sehr wichtige Aufgabe: sie stellt die Datenbankverbindung bereit und liefert EntityManager um mit Entitäten zu arbeiten.

7.7.2 PruefplankalenderHandler

Diese Klasse stellt die Liste mit den aussortierten Prüfplaneinträgen bereit. Im Folgendem werden die einzelnen Funktionen genau erläutert.

findPPDates/findPPDatesGC

Liefern nach Übergabe der Zeit, des Tages und der Woche ein Datum, entweder als String oder als GregorianCalendar-Instanz zurück.

updateMinMaxDate

Aktualisiert die Tage der Prüfperiode, die beim Eintragen einer Prüfung zur Verfügung stehen.

editPPE/newEmptyPPE

Initiiert die Maske mit Werten einer Prüfung oder „nulls“ in der Prüfplantabelle.

newPPE

Initiiert die Maske mit Werten einer Prüfung oder „nulls“ im Kalender.

updateWeek/updateBothWeeks

Aktualisieren die Kalenderwochen nach jeder Änderung der Einträge.

findPPEByTimeAndDay/findPPEByTimeAndDayString

Findet die Prüfungen für einzelne Zellen des Kalenders.

findConflicts

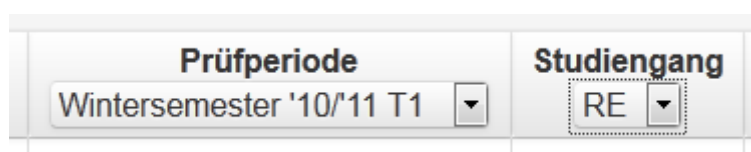
Sucht beim Eintragen einer Prüfung nach Überschneidungen.

switchPPE

Wechselt zur ausgewählten Prüfung aus der Liste der Konflikte.

7.7.3 PruefplantabelleHandler

An dieser Stelle solle auf die Filter-Funktion näher eingegangen werden, die mit DataTable realisiert wurde.



The image shows a user interface for filtering an exam schedule. It consists of two adjacent dropdown menus. The first menu is labeled 'Prüfperiode' and currently displays 'Wintersemester '10/'11 T1'. The second menu is labeled 'Studiengang' and currently displays 'RE'. Both menus have a small downward-pointing arrow on the right side of the selected text.

Abbildung 6: Prüfplantabelle: Filter

Um die Einträge zu filtern, müssen diese nicht direkt aus der Datenbank kommen, sondern können in einer Liste zwischen gespeichert sein. Außerdem benötigt man eine Select-Liste in der die Filter-Optionen sind. Diese Liste ist ein Array von SelectItem's, eine Klasse der PrimeFaces. Bei **p:dataTable** muss man angeben, wo die gefilterten Ergebnisse zu speichern sind und bei **p:column** - wonach die Spalte gefiltert werden soll (filterBy-Attribut). Wenn alles richtig eingestellt und initiiert wurde, lässt sich die Tabelle nach gewünschten Optionen filtern.

7.7.4 PrintHandler

Dies ist eine einfache Klasse, die viele getter-Methoden vom Typ String hat. In dieser Klasse werden Inhalte des Prüfplanes generiert und formatiert.

7.8 Login/Logout

Der Klasse **SessionManager** ist für die Sessionverwaltung, also für Login und Logout zuständig. Beim Login wird die Datenbank nach einem Benutzer mit dem Namen und Password von der Login-Seite gesucht. Dies geschieht in der Methode login(). Der Rückgabewert dieser Methode ist ein String, in dem die Weiterleitungsadresse steht. Im Erfolgsfall wird man auf die Übersichtsseite weitergeleitet und der eingeloggte Benutzer in der Session gespeichert. Wenn der Benutzer nicht gefunden wird, wird man auf die Login-Seite weitergeleitet. Beim Logout wird der angemeldete Benutzer aus der Session gelöscht, die Session selbst wird gelöscht und man wird auf die Login-Seite weitergeleitet.

Außerdem liefert die Methode die Navigations- und Übersichtsinhalte je nach Benutzergruppe des eingeloggten Benutzers zurück.

7.8.1 Login-Filter

Logout alleine würde den ausgeloggten Benutzer nicht davon abhalten, mit dem Zurück-Button des Browsers die besuchten Seiten aufzurufen oder durch das Eintippen eines Pfades in der Adressleiste, auf gewünschte Seite zuzugreifen. Diese Aufgabe übernimmt der Login-Filter. Hier der Code dieser Klasse:

LoginFilter.java

```
1 import ...
2
3 @WebFilter("/app/*")
4 public class LoginFilter implements Filter {
5     ...
6     @Override
7     public void doFilter(ServletRequest request,
8         ServletResponse response,
9         FilterChain chain)
10         throws ServletException, IOException {
11         HttpServletRequest req = (HttpServletRequest) request;
12         SessionManager auth =
13         (SessionManager) req.getSession().getAttribute("sessionManager");
14
15         if (auth != null && auth.isLoggedIn()) {
16             chain.doFilter(request, response);
17         } else {
18             HttpServletResponse res = (HttpServletResponse) response;
19             res.sendRedirect(req.getContextPath() + "/login.xhtml");
20         }
21     }
22     ...
23 }
```

Dank Annotationen muss der Filter nicht separat in der web.xml registriert werden. Der Filter wird bei jeder HTTP-Anfrage aufgerufen und überprüft die Session auf Vorhandensein eines eingeloggten Benutzers. Wenn dies der Fall ist, so wird die Anfrage ganz normal ausgeführt, ansonsten wird man auf die Login-Seite weitergeleitet.

Bei der Filter-Annotation wird angegeben, für welche Dateien dieser Filter gilt. Hierbei würde / bedeuten, dass die komplette Anwendung geschützt sein soll. Beim Aufruf der Login-Seite im ausgeloggten Zustand gibt es keinen eingeloggten Benutzer, somit würde diese Aktion zu einem Seitenladefehler führen und man würde auf die Login-Seite weitergeleitet werden. Dort würde wiederum der Filter aufgerufen werden usw.

Deswegen sind die Dateien, die mit dem Filter geschützt sein sollen in den Ordner **app** ausgelagert worden. Dies ist das einfachste Mechanismus eine Anwendung vom fremden Zugriff zu schützen und könnte bei einer Weiterentwicklung verbessert werden.

Teil III

Zusammenfassung und Ausblick

8 Zusammenfassung

Als ich mit der Studienarbeit angefangen habe, hatte ich noch keinerlei Erfahrung mit Java EE gehabt. Der Anfang war deswegen schwer: man musste viel nachlesen, viel ausprobieren und oft einige Tage nach Lösung eines winzigen Problems suchen. Doch ist mir der Einstieg in die Thematik gelungen und jetzt fühle ich mich sogar komplizierteren und umfangreicheren Projekten auf diesem Gebiet gewachsen.

JSF ist sehr von HTML abstrahiert und nimmt einem, vor allem unerfahrenen Entwickler sehr viel Arbeit ab. Man muss sich kaum um Positionierung, Größe oder Maßstab der Elemente kümmern, genau so wenig wie ums Design. Das schwierigste war zu verstehen, wie eine JSF-Seite mit einer Java-Klasse kommuniziert. Nachdem der Managed-Bean-Mechanismus deutlich wurde, hatte man schon den größten Teil der Arbeit hinter sich. Einen eigenen Converter zu schreiben war der nächste Knackpunkt. Obwohl die Klasse so einfach zu sein scheint, hat es mindestens eine Woche gedauert bis man den Ansatz hatte.

Die Selbstgestaltung der Elemente ist ein Kritikpunkt an JSF und vor allem an PrimeFaces. Dieser Code zeigt welche Klasse in dem Stylesheet definiert werden musste, damit der Footer schmaler ist und einen schwarzen Hintergrund hat:

Stylesheet: Footer

```
1 .footer, .footer .ui-layout, .footer .ui-layout-bd, .footer
.ui-layout-unit-content {
2     background-color: #333 !important;
3     color: #FFF;
4 }
```

Wenn man den Einsatz von JSF-Frameworks vermeidet, so ist man gezwungen viele Sachen selbst zu implementieren, was schon mal nicht optimal ist. Dazu kommt noch, dass manche Sache aus irgendeinem Framework genommen werden müssen, wie z.B. DatePicker, denn so etwas selbst zu implementieren ist erstens viel zu viel Aufwand und zweitens nicht Ziel dieser Arbeit ist. Da bietet es sich an, bei kleineren Projekten JSP mit Struts und jQuery einzusetzen, weil diese Kombination mehr Flexibilität bietet und die Anwendung trotzdem übersichtlich bleibt.

Ist man aber mit dem Design von PrimeFaces einverstanden und möchte dieses nichts großartig ändern, so ist man bei diesem Framework sehr gut aufgehoben. Allein das DataTable bringt schon so viele Funktionen und Eigenschaften mit, wie Instant row selection, Sortier- und Filterfunktionen und Export, dass es sich lohnt. Und mit dem Layout braucht man sich keine Gedanken darüber machen wie groß die Steuerelemente sein sollen und wo diese zu platzieren sind.

Die Entwicklungsumgebung und insbesondere NetBeans hat sich als sehr benutzerfreundlich und intelligent erwiesen. Die Autovervollständigung, Codegenerierung und eingebauter GlassFish Applikation-Server, der nicht konfiguriert werden muss, sind hier sehr gut realisiert und meiner Meinung nach auch viel besser als bei der Alternative: Eclipse.

Fertige Java EE Anwendungen, werden in einer .war-Datei verpackt und können über intuitive GlassFish Admin-Konsole, die über Web-Browser erreichbar ist, einfach auf den Server geladen werden.

9 Ausblick

Die wichtigsten Funktionen der Prüfplanverwaltung sind in diesem Projekt zwar implementiert worden, aber einige zusätzliche, dem Prüflanersteller das Leben-erleichternde Funktionen, sind noch nicht vorhanden. Diese Funktionalitäten sind in den Abgrenzungskriterien zu finden und sollten bei einer Weiterentwicklung implementiert werden. Der Vorschlag von Herrn Nimz, dem Entwickler der PHP-basierten Lösung, über die Verknüpfung von Studiengangübergreifenden Fächer, sollte in Zukunft berücksichtigt werden. Des Weiteren gibt es andere Optimierungsvorschläge:

- **Verknüpfung von Personen mit Accounts**

Die Verbindung zwischen Mitarbeiter/Prüfer und Account sollte aufgebaut sein, denn momentan gehören die Accounts nicht zu den wirklichen Objekten dieser Tabellen.

- **Password**

Passwortänderung/-vergabe sollte geändert werden. Zur Zeit wird beim Anlegen eines Accounts ein Passwort erwartet und falls dieses nicht eingegeben wird, wird standardmäßig 123 vergeben. Beim Ändern eines Accounts kann man ein neues Passwort eingeben ohne das alte eingegeben zu haben. Da es keine Passwortwiederherstellungsmöglichkeit gibt, ist es auch in Ordnung, denn falls ein Passwort vergessen wurde, kann er leicht ersetzt werden ohne das alte zu wissen.

- **Registrierung**

In Zukunft könnte man die Registrierung „professioneller“ gestalten (zur Zeit gibt es gar keine, außer der Administrator legt einen Benutzer ein). Man könnte sich z.B. mit einer FH-Email registrieren und an diese Email-Adresse würde man auch das Passwort aus dem vorherigen Punkt senden.

- **Rückmeldung**

Die Rückmeldung an den Benutzer, die durch **p:growl** umgesetzt worden ist, erscheint nicht in allen Seiten. Der Grund dafür ist die unmittelbare Filterung nachdem dem Neuladen der Inhalte der Tabelle.

- **Drucken**

Zur Zeit gelingt es nicht mit der Druck-Funktion des Browsers den Prüfungsplan zu drucken. Der Grund dafür ist, dass die Steuerelemente wie Header, Footer und Navigation sich über ein Print-Stylesheet zwar verstecken lassen aber den Platz, an dem sie waren, trotzdem belegen. Es wäre also denkbar dieses Problem in Zukunft zu beheben.

- **Prüfcodes**

Da ich nicht ganz die Verknüpfung zwischen Modulen und Prüfcodes verstehe, ist es mir nicht gelungen diesen Bereich zu optimieren. In der PHP-Version kann man beim Anlegen eines Moduls einen Prüfcode vergeben. Diese Information wird aber nirgendwo gespeichert, also habe ich in dieser Version dieses Feld weggelassen. Bei den Prüfcodes kann man neue Prüfcodes anlegen und diesen dann Modulen zuweisen, und zwar ein Modul pro Prüfcode. In der Datenbank findet man aber öfters 2-3 Prüfungen mit dem gleichen Prüfcode, und das sind teilweise unterschiedliche Prüfungen. Es kann sein, dass meine Datenbank nicht intakt ist.

Die Anwendung lässt also noch viel Raum für die Optimierung, aber das Ziel war ja auch eine Basis für diese Optimierung zu schaffen.

Teil IV

Anhang

Tabellenverzeichnis

1	Entity Annotationen	8
2	Gültigkeitsbereiche (Scope)	16

Abbildungsverzeichnis

1	Prüfplankalender	20
2	Überschneidungen	20
3	Modulauswahl	21
4	Prüfperiode im Kalender	21
5	Eintrag trotz Überschneidungen	22
6	Prüfplantabelle: Filter	28

Literatur

- [1] <http://itroman.wordpress.com/2010/12/05/why-i-do-hate-hibernate/>
- [2] <http://softaria.com/2011/07/26/eclipselink-versus-hibernate/>
- [3] http://www.theserverside.com/news/thread.tss?thread_id=53142
- [4] http://en.wikipedia.org/wiki/Entity_Bean
- [5] <http://wiki.eclipse.org/EclipseLink/Examples/JPA/Dynamic>
- [6] <http://java.boot.by/scbcd5-guide/ch05s07.html>
- [7] <http://www.objectdb.com/api/java/jpa>
- [8] http://jsfatwork.irian.at/book_de/jsf.html#!idx:/jsf.html:sec:custom-converters
- [9] <http://primefaces.org/>
- [10] <http://www.primefaces.org/showcase/ui/layoutHome.jsf>
- [11] <http://www.primefaces.org/showcase/ui/menu.jsf>
- [12] <http://www.primefaces.org/showcase/ui/dialogHome.jsf>
- [13] <http://www.primefaces.org/showcase/ui/datatableHome.jsf>
- [14] <http://www.primefaces.org/showcase/ui/calendarHome.jsf>
- [15] <http://www.primefaces.org/showcase/ui/printer.jsf>
- [16] <http://www.primefaces.org/showcase/ui/selectOneMenu.jsf>
- [17] <http://stackoverflow.com/questions/4815722/jsf-vs-facelets-vs-jsp>
- [18] <http://forum.primefaces.org/>