

Kurzfassung

Die gegenständliche Arbeit beschreibt die Implementierung einer Web-Plattform, die im Fernunterricht zum Einsatz kommen soll. Ihre Aufgabe liegt in der Durchführung von Umfragen, die während einer Online Lehreinheit abgehalten werden. Die Zusammenstellung dieser Umfragen soll einerseits manuell über die Anwendung erfolgen, andererseits soll das Question & Test Interoperability (QTI) Format für den Import unterstützt werden. Letzteres ist ein standardisiertes Format, welches bereits in zahlreichen Anwendungen zum Einsatz kommt. Es bietet zahlreiche vorgefertigte Fragetypen an, mit deren Hilfe sich auch komplexe Umfragen, zum Beispiel mit Verzweigungen, erstellen lassen.

Abstract

This master thesis describes the implementation of a web platform, which shall be used during teaching lessons. The main task is the execution and evaluation of online surveys. On the one hand the composition of questions using the web application should be possible, on the other hand the Question & Test Interoperability (QTI) format should be supported. QTI is an upcoming format, which is already used in several projects. It provides various different question types, which can be composed to complex surveys, e.g. by using branches.

Danksagung

An dieser Stelle möchte ich mich bei all den Menschen bedanken, die mir diese Arbeit ermöglicht haben.

Besonderer Dank gilt meinem Betreuer Herrn Priv.-Doz. Mag. Dipl.-Ing. Dr. Michael Sonntag. Er hat mir von Anfang an ein klares Ziel vorgegeben und mich mit nützlichen Hinweisen in die richtige Richtung dirigiert.

Weiters möchte ich mich bei Herrn Dr. Jonathon Hare von der University of Southampton bedanken. Ohne seine Hilfe hätte sich die Integration des QTI Frameworks bei weitem schwieriger gestaltet.

Natürlich sollen an dieser Stelle meine Eltern Reinhold und Anna Prinz nicht unerwähnt bleiben. Ihnen gebührt ein besonderer Dank, da sie mir das Studium finanziell ermöglicht und mich stets zum Weitermachen motiviert haben.

Ein grosses Danke auch an meine Freunde und Kollegen, die mich während dem Studium begleitet haben und mir mit Rat und Tat zur Seite gestanden sind.

Zu guter Letzt möchte ich mich noch bei Andrea bedanken. Sie hat mir im letzten Jahr immer die nötige Kraft und Ausdauer verliehen und mich ermutigt wenn etwas nicht nach Plan lief.

Inhaltsverzeichnis

1	Einleitung	15
1.1	Anforderungen	15
1.2	Question & Test Interoperability	16
1.3	Aufbau einer Frage im QTI Format	22
1.4	Anwendungen mit QTI Unterstützung	25
1.4.1	WeLearn	25
1.4.2	Online Learning And Training (OLAT)	26
1.4.3	QuestionMark Perception	26
1.4.4	ILIAS	26
1.4.5	Moodle	27
1.4.6	Sakai	27
1.4.7	KUSSS	27
2	Education Platform	29
2.1	Die Funktionalität	29
2.1.1	Funktionen der Anwendung	29
2.1.2	Wozu wird die Anwendung verwendet?	31
2.1.3	Verwendung von AJAX	31
2.2	Verwendete Technologien	32
2.2.1	Java	32
2.2.2	Eclipse	33
2.2.3	JavaServer Faces	33
2.2.4	RichFaces	35
2.2.5	Facelets	36
2.2.6	Hibernate	37
2.2.7	Datenbank: MySQL oder HSQLDB	40
2.2.8	Ant	40

2.2.9	Tomcat	42
2.2.10	XML	42
2.2.11	Log4j	43
2.2.12	Zusammenfassung	43
2.3	QTI	44
2.3.1	QTITools	44
2.3.2	Die Kommunikation	45
2.3.3	Anpassungen	49
2.4	Die Implementierung	49
2.4.1	Struktur der Anwendung	49
2.4.2	Aufbau der Ansicht mit Facelets	51
2.4.3	Webseiten, integrierte Komponenten und AJAX	53
2.4.4	Geschäftslogik	60
2.4.5	Datenbank in Verbindung mit Hibernate	63
2.4.6	Rollen und Rechte	65
2.4.7	Logging	66
2.4.8	Filter	67
2.4.9	Einstellungen durch Properties	68
2.4.10	Stylesheets mit CSS	69
2.5	Die Datenbank	69
2.6	Testen der Anwendung	70
2.6.1	Kompatibilitätstests	71
2.6.2	Kommunikation	71
2.6.3	AJAX-Komponenten	71
2.6.4	Filter	73
2.6.5	QTI Player	73
2.6.6	Datenbankzugriffe	74
2.6.7	Rollen und Rechte	75
2.7	Probleme während der Entwicklung und deren Lösungen	75
2.7.1	Technologische Probleme	76
2.7.2	Implementierungstechnische Probleme	77
2.7.3	QTI Probleme	79
3	Installations- und Bedienungsanleitung	81
3.1	Installation	81

3.1.1	Datenbank	82
3.1.2	Tomcat	82
3.1.3	QTI playr	83
3.1.4	Anwendung	83
3.2	Bedienungsanleitung	84
3.2.1	Use Case 1 - Umfrage manuell oder auf Basis eines QTI Content Package erstellen	84
3.2.2	Use Case 2 - Durchführung einer Umfrage aus der Sicht eines Lehrers	88
3.2.3	Use Case 3 - Teilnahme an einer Umfrage aus der Sicht eines Stu- denten	89
3.2.4	Use Case 4 - Teilnahme an einer anonymen Umfrage aus der Sicht eines Studenten	91
3.2.5	Use Case 5 - Benutzeradministration	93
4	Erweiterungsmöglichkeiten	95

Abbildungsverzeichnis

1.1	Choice	17
1.2	Associate	17
1.3	Inline Choice	18
1.4	Extended Text	18
1.5	Hotspot	18
1.6	Gap Match	19
1.7	Graphic Associate	19
1.8	Position Object	19
1.9	Order	20
1.10	Match	20
1.11	Text Entry	20
1.12	Hot Text	20
1.13	Graphic Gap Match	20
1.14	Select Point	21
1.15	Graphic Order	21
1.16	Slider	21
1.17	Ablauf eines assessmentTest	25
2.1	JSF Lifecycle [JSF]	34
2.2	Hibernate Lifecycle [HIBPER]	39
2.3	ASDEL [ASDEL1]	46
2.4	R2Q2 [ASDEL2]	47
2.5	Kommunikationsablauf [QTITOOLS]	48
2.6	Verzeichnisstruktur	50
2.7	Seite mit leerem Hauptteil	53
2.8	RichFaces Progressbar	58
2.9	RichFaces Upload	59

3.1	Login	84
3.2	Umfragen Übersicht	85
3.3	Schritt 1 - Name der Umfrage eingeben	85
3.4	Schritt 2 - Frage hinzufügen	86
3.5	Schritt 3 - Frage festlegen	86
3.6	Schritt 4 - Label definieren	86
3.7	Schritt 5 - Umfrage speichern oder neue Frage hinzufügen	87
3.8	Laden einer QTI Umfrage	87
3.9	Logout	88
3.10	Schritt 1 - Umfrage zuweisen	88
3.11	Schritt 2 - Studenten melden sich an	89
3.12	Schritt 3 - Fortschritt der Umfrage beobachten	89
3.13	Schritt 1 - Auf Umfrage warten	90
3.14	Schritt 2 - Bei Umfrage anmelden	90
3.15	Schritt 3 - Umfrage starten	90
3.16	Schritt 4 - Selbst erstellte Umfrage beantworten	90
3.17	Schritt 4 - QTI Umfrage beantworten	91
3.18	Schritt 1 - Auf Umfrage warten	91
3.19	Schritt 2 - Bei Umfrage anmelden	92
3.20	Schritt 3 - Umfrage starten	92
3.21	Schritt 4 - Selbst erstellte Umfrage beantworten	92
3.22	Schritt 4 - QTI Umfrage beantworten	93
3.23	Benutzer Übersicht	93

Tabellenverzeichnis

2.1	Überblick über die verwendeten Technologien	43
2.2	Webseiten-Kategorien	54
2.3	RichFaces Komponenten	60
2.4	Packages und Java Klassen	61
2.5	Datenbanktabellen	70
2.6	Methoden und Resultate des QTIPlayer	74

Abkürzungsverzeichnis

AJAX	Asynchronous JavaScript and XML
CSS	Cascading StyleSheets
GPL	General Public Licence
HQL	Hibernate Query Language
HTML	HyperText Markup Language
JDBC	Java DataBase Connectivity
JSF	JavaServer Faces
JSP	JavaServer Pages
LMS	Learning Management System
QML	Questions Markup Language
QTI	Question and Test Interoperability
UML	Unified Modelling Language
XML	eXtensible Markup Language
XSLT	eXtensible Stylesheet Language Transformation

Kapitel 1

Einleitung

Im Rahmen dieses Kapitels sollen zuerst die Anforderungen an die entwickelte Anwendung genauer erläutert werden. Weiters wird das Thema Question & Test Interoperability (QTI) angesprochen. Dazu wird in einem ersten Abschnitt erklärt, was man sich unter dem Begriff QTI vorstellen darf. Im nächsten Schritt werden die Bestandteile eines QTI Tests aufgelistet und beschrieben, besonderes Augenmerk liegt dabei auf dem Aufbau der diversen XML Dateien. Den Abschluss bildet eine beispielhafte Aufzählung von Anwendungen die bereits QTI unterstützen mit einer kurzen Beschreibung von Einsatzgebieten.

1.1 Anforderungen

Es soll eine Plattform entwickelt werden, mit deren Hilfe der Vortragende einer Lehrveranstaltung den Unterricht interaktiver gestalten kann. Während einer solchen Lehrveranstaltung sollen in erster Linie Live-Online-Umfragen durchgeführt werden können und deren Resultate als Feedback für den Vortragenden und die Studenten dienen. Wichtig ist dabei auch die ständige Aktualisierung der Daten des Vortragenden, so dass er immer einen Überblick über die Resultate seiner Teilnehmer behält. Im Falle einer Prüfung soll der Leiter mehrere Steuerelemente zur Verfügung haben, etwa eine Anzeige der bisher beantworteten Fragen, einen Button für den sofortigen Stop aller derzeit laufenden Tests/Umfragen oder einen Link, der auf eine Resultatseite führt, auf der die Ergebnisse sowie generelle Statistiken angezeigt werden können. Weiters soll es mehrere

Alternativen der Anmeldung am System geben, einerseits einen Administrationszugang für den Leiter, andererseits einen Login für den Teilnehmer, welcher sowohl anonym als auch persönlich sein kann. Des Weiteren sollten einige technische Details berücksichtigt werden. Bei der Plattform soll es sich nicht um eine Swing-Anwendung oder Ähnlichem handeln, sondern eine Web-Anwendung muss implementiert werden. Die Funktionalität hat dabei eine höhere Priorität als die Ansehnlichkeit der Oberfläche. Im Hintergrund soll eine Datenbank laufen, in die alle Daten eingepflegt werden. Das Programm soll so gestaltet werden, dass es nicht an eine bestimmte Datenbank gebunden ist, sondern bei Bedarf auch mit anderen Datenbanken zusammenarbeiten kann.

1.2 Question & Test Interoperability

Question & Test Interoperability, auch bekannt unter der Abkürzung QTI, wurde im Frühjahr 2000 vom IMS Global Learning Consortium offiziell als Standard definiert. Auf der offiziellen Homepage findet man zu ihm folgende Erklärung [IMS]:

The IMS Question & Test Interoperability (QTI) specification describes a data model for the representation of question (assessmentItem) and test (assessmentTest) data and their corresponding results reports. Therefore, the specification enables the exchange of this item, test and results data between authoring tools, item banks, test constructional tools, learning systems and assessment delivery systems. The data model is described abstractly, using UML to facilitate binding to a wide range of data-modelling tools and programming languages, however, for interchange between systems a binding is provided to the industry standard XML and use of this binding is strongly recommended. The IMS QTI specification has been designed to support both interoperability and innovation through the provision of well-defined extension points. These extension points can be used to wrap specialized or proprietary data in ways that allows it to be used alongside items that can be represented directly.

Was darf man sich nun unter dieser Definition vorstellen? Im Prinzip handelt es sich um eine komplexe Beschreibung von Fragen und Tests in abstrakter Form (Datenmodell). Eine genauere syntaktische Beschreibung davon ist für XML spezifiziert und folgt im

nächstes Abschnitt. Zunächst stellt sich aber die Frage, wozu dieses Format entwickelt wurde. Das IMS Global Consortium hat nach einem allgemeinen Format für Fragen gesucht, das nicht speziell auf ein Anwendungsgebiet zugeschnitten war. Ein weiterer Motivationsfaktor war die Zeitersparnis, die ein wiederverwendbares Format mit sich bringen würde. Während zuvor jede Anwendung ihr eigenes Format einsetzte, konnte nun diese Entwicklungszeit eingespart werden, indem man bereits existierende Fragen heranzog. Die erste öffentliche Version (1.0) wurde im Frühjahr 2000 vorgestellt. Diese Version bediente sich zu dem Zeitpunkt noch der Sprache QML der Firma Question-Mark. Im Laufe der Jahre entwickelte sich die Sprache weiter und wurde schliesslich in XML umgesetzt. Mit der heutigen Version 2.0 werden diese 16 Fragetypen angeboten [IMS_TYPES]:

Choice: Auswahl von einer oder mehreren Antwortmöglichkeiten

UNATTENDED LUGGAGE

Look at the text in the picture.

**NEVER LEAVE
LUGGAGE
UNATTENDED**

What does it say?	
You must stay with your luggage at all times.	<input type="radio"/>
Do not let someone else look after your luggage.	<input type="radio"/>
Remember your luggage when you leave.	<input type="radio"/>

Abbildung 1.1: Choice

Associate: Gegebene Begriffe müssen ausgewählt und einander (paarweise) zugeordnet werden

SHAKESPEARIAN RIVALS

Hidden in this list of characters from famous Shakespeare plays are three pairs of rivals. Can you match each character to his adversary?

Lysander	Prospero
Antonio	
Capulet	Montague

—
 —
 —

Abbildung 1.2: Associate

Inline Choice: Im Lückentext müssen Begriffe aus einer Auswahl selektiert werden

RICHARD III (TAKE 2)

Identify the missing word in this famous quotation from Shakespeare's Richard III.

Now is the winter of our discontent
Made glorious summer by this sun of ;
And all the clouds that lour'd upon our house
In the deep bosom of the ocean buried.

Abbildung 1.3: Inline Choice

Extended Text: Ein eigener Text mit begrenzter Zeichenanzahl muss eingegeben werden

WRITING A POSTCARD

Read this postcard from your English pen-friend, Sam.

Here is a postcard of my town. Please send me a postcard from your town. What size is your town? What is the nicest part of your town? Where do you go in the evenings?

Sam.

Write Sam a postcard. Answer the questions. Write 25-35 words.

Abbildung 1.4: Extended Text

Hotspot: In einer Grafik soll aus mehreren markierten Positionen eine bestimmte ausgewählt werden

UK AIRPORTS (TAKE 1)

The picture illustrates four of the most popular destinations for air travellers arriving in the United Kingdom: London, Manchester, Edinburgh and Glasgow.

Which one is Glasgow?

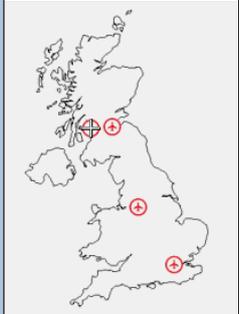


Abbildung 1.5: Hotspot

Gap Match: Im Lückentext müssen Begriffe aus einem Pool korrekt zugeordnet werden

RICHARD III (TAKE 1)

Identify the missing words in this famous quotation from Shakespeare's Richard III.

Now is the winter of our discontent
 Made glorious Word 2 by this sun of York;
 And all the clouds that lour'd upon our house
 In the deep bosom of the ocean buried.

Use the table below to select the missing words.

	winter	spring	summer	autumn
Word 1	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Word 2	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Abbildung 1.6: Gap Match

Graphic Associate: Markierte Positionen in einer Grafik sollen einander (paarweise) zugeordnet werden

LOW-COST FLYING

Frizz, a new low cost airline, already operates a service connecting Manchester and Edinburgh but has recently opened two new routes: a service between London and Edinburgh and one between London and Manchester.

Mark the airline's new routes on the airport map:



Drag the markers by their ends to connect the appropriate points on the image

Abbildung 1.7: Graphic Associate

Position Object: Ein Objekt soll in einer Grafik positioniert werden

AIRPORT LOCATIONS

When flying into the UK, you may well find yourself landing at Edinburgh, Manchester or London Heathrow; but where are these airports actually located?



Abbildung 1.8: Position Object

Order: Begriffe müssen in eine korrekte Reihenfolge gebracht werden

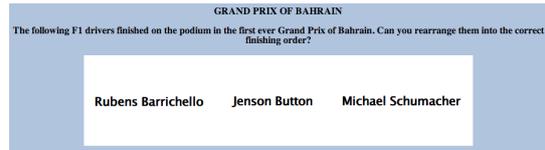


Abbildung 1.9: Order

Match: Zu gegebenen Begriffen müssen andere zugeordnet werden

CHARACTERS AND PLAYS			
Match the following characters to the Shakespeare play they appeared in:	The Tempest	Romeo and Juliet	A Midsummer-Night's Dream
Prospero	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Capulet	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Demetrius	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Lysander	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Abbildung 1.10: Match

Text Entry: Im Lückentext müssen Wörter eigenständig eingetragen werden

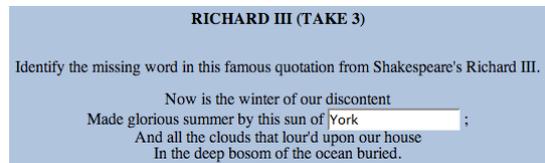


Abbildung 1.11: Text Entry

Hot Text: Im Text soll entschieden werden, ob zur Auswahl stehende Wörter an dieser Position fehlen

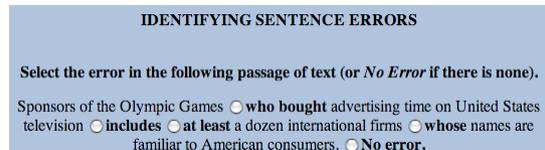


Abbildung 1.12: Hot Text

Graphic Gap Match: Gegebene Begriffe sollen in einer Grafik markierten Positionen zugeordnet werden

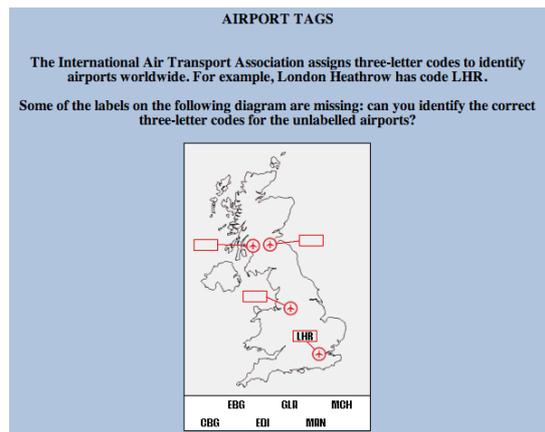


Abbildung 1.13: Graphic Gap Match

Select Point: In einer Grafik soll eine bestimmte Position möglichst genau markiert werden

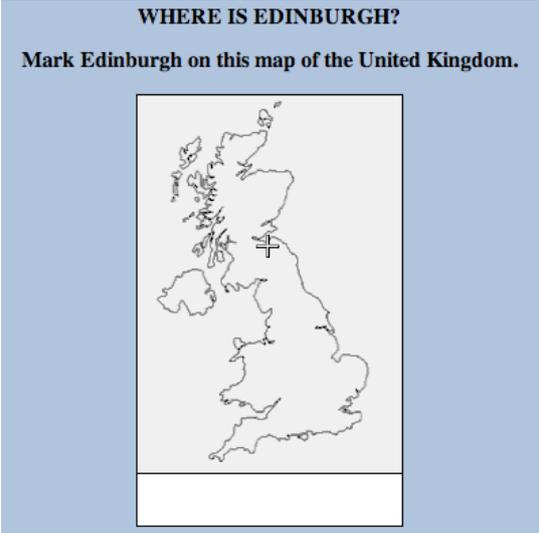


Abbildung 1.14: Select Point

Graphic Order: Die in einer Grafik angegebenen Positionen sollen in eine Reihenfolge gebracht werden

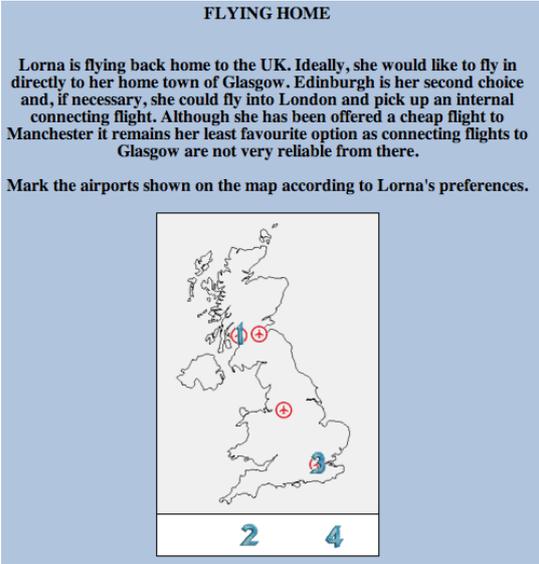


Abbildung 1.15: Graphic Order

Slider: Auf einem Schieberegler soll ein korrekter Wert eingestellt werden

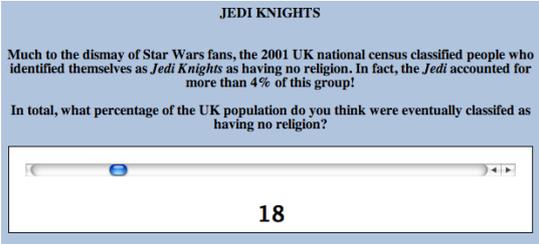


Abbildung 1.16: Slider

Version 2.1 befindet sich derzeit noch im Draft-Status, eine finale Version wird für die nahe Zukunft erwartet.

1.3 Aufbau einer Frage im QTI Format

Wie bereits erwähnt, bildet XML die Repräsentation von QTI Fragen. Dabei wird zwischen Test (*assessmentTest*) und Fragen (*assessmentItem*) unterschieden. Ein Test setzt sich aus einer oder mehreren Fragen zusammen. Listing 1.1 zeigt den Aufbau eines *assessmentItem*, d.h. einer einzelnen Frage (hier: Choice)

```
1 <assessmentItem identifier="choice" title="Auswahlmöglichkeiten">
2   <responseDeclaration identifier="RESPONSE" cardinality="multiple" baseType="identifier">
3     <correctResponse>
4       <value>ChoiceC</value>
5       <value>ChoiceA</value>
6     </correctResponse>
7   </responseDeclaration>
8   <outcomeDeclaration identifier="SCORE" cardinality="single" baseType="integer">
9     <defaultValue>
10      <value>0</value>
11    </defaultValue>
12  </outcomeDeclaration>
13  <itemBody>
14    <p>Bitte wählen Sie entweder die erste oder letzte Antwortmöglichkeit</p>
15    <choiceInteraction responseIdentifier="RESPONSE" shuffle="true" maxChoices="1">
16      <prompt>Which item is correct?</prompt>
17      <simpleChoice identifier="ChoiceA" fixed="false">Antwort A</simpleChoice>
18      <simpleChoice identifier="ChoiceB" fixed="false">Antwort B</simpleChoice>
19      <simpleChoice identifier="ChoiceC" Fixed="True">Antwort C</simpleChoice>
20    </choiceInteraction>
21  </itemBody>
22  <responseProcessing>
23    <responseCondition>
24      <responseIf>
25        <member>
26          <variable identifier="RESPONSE"/>
27          <correct identifier="RESPONSE"/>
28        </member>
29      <setOutcomeValue identifier="SCORE">
```

```

30     <baseValue baseType="integer">1</baseValue>
31   </setOutcomeValue>
32 </responseIf>
33 <responseElse>
34   <setOutcomeValue identifier="SCORE">
35     <baseValue baseType="integer">0</baseValue>
36   </setOutcomeValue>
37 </responseElse>
38 </responseCondition>
39 </responseProcessing>
40 </assessmentItem>

```

Listing 1.1: assessmentItem

Das Wurzelement *assessmentItem* bildet den Ausgangspunkt der Frage und wird durch den *identifier* eindeutig definiert. Im Element *responseDeclaration* werden die korrekten Antwortmöglichkeiten aufgelistet. Die erreichte Punkteanzahl wird im Element *outcomeDeclaration* festgelegt und initial mit einem Wert (0) versehen. Innerhalb des *itemBody* Elements befindet sich die eigentliche Frage, die mittels HTML Befehlen ansehnlich gestaltet werden kann. Es wird dafür vorausgesetzt, dass die Frage in einem Webbrowser betrachtet wird, ansonsten kann die Verwendung von HTML Befehlen zu einer fehlerhaften Darstellung führen. Die Darstellung des Fragetyps übernimmt das Element *choiceInteraction*, in diesem Fall wurde *simpleChoice* gewählt. Den Abschluss bildet das Element *responseProcessing*. Dieser Bereich ist für die Punktezuteilung verantwortlich und wird durch Bedingungen (*responseCondition*) realisiert. Hat der Benutzer die Frage richtig beantwortet bekommt er einen, ansonsten null Punkte.

Listing 1.2 entstammt einem Test von geringer Komplexität und soll die Funktionsweise eines *assessmentTest* verdeutlichen.

```

1 <assessmentTest identifier="SCENARIO" title="Szenario">
2   <testPart identifier="part1" navigationMode="linear" submissionMode="individual">
3     <itemSessionControl showFeedback="true"/>
4     <assessmentSection identifier="sectionA" title="Section_1" visible="true">
5       <assessmentItemRef identifier="item1" href="item1.xml">
6         <branchRule target="item2_2">
7           <match>
8             <variable identifier="item1.outcome"/>
9             <baseValue baseType="identifier">C</baseValue>
10          </match>

```

```

11     </branchRule>
12 </assessmentItemRef>
13 <assessmentItemRef identifier="item2_1" href="item2.xml">
14   <branchRule target="EXIT_TEST">
15     <match>
16       <variable identifier="item1.outcome"/>
17       <baseValue baseType="identifier">A</baseValue>
18     </match>
19   </branchRule>
20   <branchRule target="EXIT_TEST">
21     <match>
22       <variable identifier="item1.outcome"/>
23       <baseValue baseType="identifier">C</baseValue>
24     </match>
25   </branchRule>
26 </assessmentItemRef>
27 <assessmentItemRef identifier="item2_2" href="item3.xml">
28 </assessmentItemRef>
29 </assessmentItem>
30 </testPart>
31 </assessmentTest>

```

Listing 1.2: assessmentTest

Das Wurzelement *assessmentTest* definiert den Einstiegspunkt eines Tests und wird durch den *identifier* klar festgelegt. Das Element *testPart* erlaubt die Untergliederung eines Tests in mehrere Subkategorien, mittels *assessmentSection* können innerhalb dieser Teile weitere Sektionen definiert werden. Die *assessmentItemRef* Elemente legen Regeln (*branchRule* und *match*) fest, die bestimmen, welche Fragen als nächstes angezeigt werden sollen. Im obigen Beispiel folgt nach erfolgreicher Lösung der ersten Frage (*item1*) die dritte Frage (*item2_2*), da das *item1* als *target* im *branchRule* das *item2_2* ausgewählt hat. Voraussetzung dabei ist die Erfüllung der Anforderung innerhalb des *match* Element. Das *item1* fordert, dass als Antwort C gewählt werden muss. Entscheidet sich der Benutzer jedoch für eine andere Antwortmöglichkeit, so wird das *target* ignoriert und mit dem nächsten *assessmentItemRef*. fortgefahren. In Abbildung 1.17 lassen sich die Abhängigkeiten sehr gut erkennen.

Zwar sieht die Spezifikation des IMS Global Consortium [IMS_SPEC] noch mehr als die hier benutzten Elemente und Attribute vor, eine Auflistung würde aber den Rahmen sprengen und ist für das Grundverständnis nicht von Nöten.

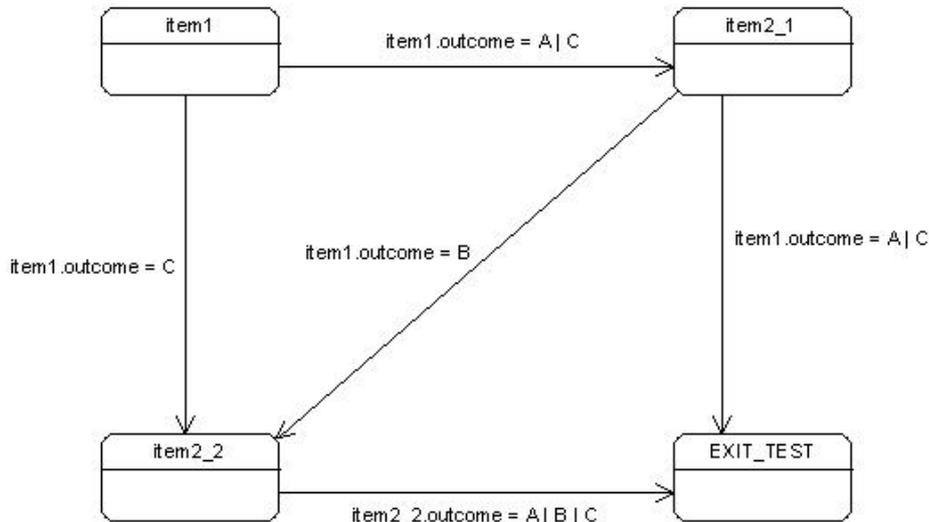


Abbildung 1.17: Ablauf eines assessmentTest

1.4 Anwendungen mit QTI Unterstützung

Viele Schulen, Universitäten und Unternehmen benutzen in der heutigen Zeit sogenannte *Learning Management Systems* (LMS) . Bei einem Learning Management System handelt es sich um eine webbasierte E-Learning Plattform, die der Bereitstellung von Lerninhalten und der Organisation von Lernvorgängen dient. Sie erleichtert den Unterricht in Schulen bzw. die Weiterbildung in Unternehmen. Meistens bieten LMS Funktionen wie Stundenplan, Benachrichtigungen, Ausführen und Bewerten von Tests und Umfragen, Darstellen von Resultaten, etc., an, die von Schülern und Studenten genutzt werden können. Je nach Entwicklungsgrad der Plattform unterscheidet sich die Funktionsvielfalt. Die folgende Aufzählung listet einige Anwendungen und Projekte auf, die sich bereits etabliert haben.

1.4.1 WeLearn

WeLearn ist eine in Java implementierte Lernplattform, die an der Johannes-Kepler-Universität Linz im Institut für Informationsverarbeitung und Mikroprozessortechnik entwickelt wurde. Neben einigen typischen Funktionen wie der Präsentation von Lehr- und Lernmaterialien oder der Kommunikation und Interaktion zwischen Lernenden und

Lehrenden wird auch der *Content Packaging Standard* des IMS Global Learning Consortiums unterstützt. Die aktuelle Version ist: WeLearn 2.3.2 - Premium Edition.

1.4.2 Online Learning And Training (OLAT)

OLAT wurde 1999 von der Universität in Zürich entwickelt. Die Open Source Webanwendung, die unter der Apache Licence 2.0 veröffentlicht wurde. Mit der Version 3.0 wurde das gesamte System mit der Programmiersprache Java von Grund auf neu gebaut. Die im September 2008 veröffentlichte Version 6.0.4 bietet alle für ein LMS typischen Funktionen und wird darüber hinaus in über 15 Sprachen angeboten [OLAT].

1.4.3 QuestionMark Perception

Die Firma QuestionMark, Mitbegründer des QTI Formats, bietet seine Anwendung unter dem Namen Perception an. Weltweit wurden bisher ungefähr 15.000 Autorensysteme installiert und Systeme für über 10 Millionen Teilnehmer eingerichtet. Das kostenpflichtige Produkt wirbt hauptsächlich mit seiner einfachen, zugänglichen und qualitativ hochwertigen Software für Durchführung von Tests. Wie auch bei anderen Anwendungen sollen damit Prüfungen, Umfragen und Auswertungen leicht von der Hand gehen [QM].

1.4.4 ILIAS

Die unter dem Akronym bekannte Anwendung ILIAS (Integriertes Lern-, Informations- und Arbeitskooperations-System) existiert bereits seit 1997. Sie wurde im Rahmen eines Projekt an der Universität Köln für interne Zwecke entwickelt. Erst im Jahr 2000 wurde ILIAS als Open Source Software unter der General Public Licence (GPL) veröffentlicht. Die aktuelle Version 3.10.1, die seit Oktober 2008 verfügbar ist, bietet eine Vielzahl an Funktionalitäten, die über das Standard Repertoire hinausgehen. [ILIAS]

1.4.5 Moodle

Das hierzulande sehr gefragte Moodle (Modular Object-Oriented Dynamic Learning Environment) kann eine beachtliche Anzahl an Benutzern weltweit aufweisen. Ebenfalls unter der GPL veröffentlicht, bietet die aktuelle Version 1.9.3 neben Standardkomponenten auch eigene innovative Funktionen. Moodle basiert auf der Programmiersprache php. [MOODLE]

1.4.6 Sakai

Das Open Source Projekt Sakai zählt zu den bekanntesten LMS und stelle eine echte Alternative zu kommerziellen System dar. Es wurde 2004 an der University of Michigan und Indiana University gestartet, das MIT und die Stanford University traten zu einem späteren Zeitpunkt bei. Mittlerweile gibt es über 80 Hochschulen, die an der Entwicklung teilhaben.

Die Software ist Java basiert und untergliedert sich in zwei Kernkomponenten, dem Sakai-Framework und den Sakai-Tools. Während das Sakai-Framework die Grundfunktionen zur Verfügung stellt, erweitern die Sakai-Tools das System um zusätzliche Funktionen. [SAKAI]

1.4.7 KUSSS

Das Kepler University Study Support System, kurz KUSSS, ist in Java geschrieben und dient den Studenten als Anlaufstelle für die Organisation der Lehrveranstaltungen. Obwohl es einige Funktionalitäten eines LMS, wie zum Beispiel das Bereitstellen von Lernmaterialien oder ein Diskussionsforum, anbietet, kann es mehr als Portal für Studenten gesehen werden. Es wird in erster Linie für die An- und Abmeldung von Lehrveranstaltungen und Prüfungen benutzt, nur wenige Lehrveranstaltungen bieten auch Unterlagen über das KUSSS an.

Kapitel 2

Education Platform

2.1 Die Funktionalität

Im Vergleich zu den bisher vorgestellten, funktionell hochwertigen LMS handelt es sich bei der implementierten Anwendung um eine sehr spezialisierte Plattform, die keine universelle Funktionalität mitbringt. Der Fokus liegt klar auf der Durchführung von Umfragen und Tests und ist für den Gebrauch im Fernunterricht in Schulen bzw. Universitäten ausgerichtet. In den folgenden Unterkapiteln werden zuerst die Funktionen der Anwendung beleuchtet, um einen groben Überblick zu geben, welcher Bereich eines Lernsystems durch die Plattform abgedeckt wird. Weiters soll anhand mehrerer Szenarien die genaue Verwendungsweise erläutert werden. Schliesslich wird noch die Rolle von AJAX in Bezug auf die Anwendung besprochen.

2.1.1 Funktionen der Anwendung

Die Anwendung konzentriert sich auf die folgenden Funktionen:

- Anmeldung am System (über Login oder anonym)
- Erstellen einer Umfrage (manuell oder auf Basis eines QTI Content Packages)
- Durchführen der Umfrage (Vortragender wird über den Fortschritt benachrichtigt und kann jederzeit abbrechen)
- Präsentation der Resultate

Bevor der Benutzer die Startseite betreten kann, muss er sich mit einem gültigen Benutzernamen und Passwort am System anmelden. Je nachdem welche Rechte dem Benutzer bei der Erstellung seines Zugangs zugeteilt wurden, sieht er auf der Startseite mehr oder weniger Interaktionsmöglichkeiten. Als Administrator stehen einem alle Funktionen zu Verfügung, es können zum Beispiel neue Benutzer angelegt oder bestehende entfernt werden.

Der Student ist nach seiner Anmeldung bereits auf wenige Funktionen begrenzt, dazu zählen das Lesen von Meldungen und die Durchführung von Umfragen/Tests. Schliesslich gibt es noch eine anonyme Anmeldung, die über einen festgelegten Link und nicht über den Login erfolgt und lediglich die Teilnahme an Umfragen/Tests zulässt.

Ist der Benutzer als Administrator oder Lehrer am System angemeldet, bietet sich eine weitere Funktionalität, der QTI Player, an. Damit lassen sich bereits erstellte Umfragen im QTI Format vorab ausführen und die Resultate betrachten. Wenn das Ergebnis zufriedenstellend ist, kann auf Basis der QTI Datei eine Umfrage erstellt werden, an der beliebig viele Personen teilnehmen können. Falls die gewünschte Umfrage nicht im QTI Format vorliegen sollte, kann ein Benutzer mit entsprechenden Rechten auch manuell einfache Fragen (d.h. nicht der volle QTI Umfang) zu einer Umfrage zusammensetzen und den Teilnehmern vorlegen. Während der Durchführung einer Umfrage stehen dem Vortragenden Kontrollmechanismen zur Verfügung. Über eine Status Anzeige wird der Fortschritt aller Teilnehmer graphisch präsentiert und es kann sowohl die gesamte Umfrage jederzeit abgebrochen werden, als auch die Umfrage eines einzelnen Teilnehmers. Nach Abschluss der Umfrage bietet sich dem Vortragenden die Möglichkeit die Resultate zu betrachten.

Wie bereits erwähnt ist das User-Management dem Administrator vorbehalten. Dieser darf jederzeit Benutzer hinzufügen oder entfernen. Weiters können Gruppen angelegt und diesen anschliessend Benutzer zugeteilt werden. Wird zu einem späteren Zeitpunkt eine Umfrage gestartet, brauchen nur mehr Gruppen und nicht jeder einzelne Benutzer für diese freigeschalten werden.

Schliesslich gilt es noch die Meldungen zu erwähnen. Meldet sich ein Benutzer als Student am System an, wird er automatisch auf die Startseite weitergeleitet. Falls der Administrator oder Lehrer eine oder mehrere Meldungen verfasst hat, werden diese dem Studenten auf der Startseite angezeigt.

2.1.2 Wozu wird die Anwendung verwendet?

Der primäre Zweck der Anwendung liegt in der Durchführung von Umfragen, deshalb wird anhand der folgenden beiden Szenarien die Verwendungsweise etwas genauer erläutert. Das erste Szenario beschreibt die Interaktion mit dem System aus der Perspektive des Lehrers, während das zweite Szenario aus der Sicht eines Schülers geschildert wird.

Szenario 1: Lehrer

- Am System als Teacher anmelden
- Eine Umfrage erstellen (manuell oder auf Basis eines QTI Content Packages)
- Die Umfrage öffnen
- Gruppe(n) zuweisen
- Den Status auf 'OPEN' setzen
- Die Umfrage starten und den Fortschritt betrachten
- Wenn nötig, die gesamte Umfrage oder die eines einzelnen Studenten abbrechen
- Umfrage abschliessen und die Resultate begutachten
- Auswerten und Diskutieren

Szenario 2: Teilnehmer

- Am System als Student anmelden
- Warten bis eine Umfrage geöffnet wird
- Für die Umfrage anmelden
- Die Fragen beantworten
- Die Umfrage abschliessen
- Warten
- Gesamtergebnis ansehen

2.1.3 Verwendung von AJAX

Wie die beiden Szenarien zeigen, ist die interne Kommunikation zwischen Lehrer und Student essentiell für die Ausführung einer Umfrage. Erreicht wird diese Kommunikation durch die Verwendung von Komponenten, die AJAX (Asynchronous JavaScript And XML) unterstützen. Die genauere Beschreibung der Komponenten folgt im nächsten

Kapitel. An dieser Stelle will aber der Gedankengang, der hinter AJAX steckt, erwähnt werden. Es handelt sich dabei um keine Web-Browser Technologie, sondern vielmehr um ein Konzept. Wie der Name Asynchronous JavaScript And XML bereits aussagt basiert das Konzept auf einer asynchronen Datenübertragung. Normalerweise wird bei synchronen Zugriffen die ganze Seite neu geladen. Das bringt zwei grosse Nachteile mit sich: Einerseits kann das zu langen Ladezeiten führen, andererseits kann bei komplexen Seiten ein Reload die Funktionalität zerstören (z.B. das Popup Fenster für die manuelle Umfrage bei einem Studenten wird nach einem Page Reload geschlossen). AJAX hingegen führt die Übertragung nur für benötigte Komponenten durch, aktualisiert die Teile entsprechend der Antwort des Servers und verhindert dadurch das erneute Laden der kompletten Seite. In Bezug auf die vorliegende Arbeit wird AJAX für die Kommunikation verwendet. Auf der einen Seite setzt der Student regelmäßig asynchrone Abfragen ab um zu ermitteln, ob eine Umfrage zur Verfügung steht. Auf der anderen Seite ermittelt der Teacher über AJAX Zugriffe wieviele Studenten sich bereits für die von ihm eröffnete Umfrage angemeldet haben. Sobald die Umfrage startet, kann der Teacher über Fortschrittsbalken beobachten, bei welcher Frage sich jeder einzelne Student im Moment befindet. Auch diese Funktionalität wird mittels AJAX realisiert, indem kontinuierlich der aktuelle Status jedes Teilnehmers abgefragt wird.

2.2 Verwendete Technologien

Dieser Abschnitt behandelt die wichtigsten Technologien, die in der Implementierung verwendet werden. Es soll dabei ein Überblick über die verwendete Version, den Zweck und die Voraussetzungen (falls vorhanden) der Technologie gegeben werden.

2.2.1 Java

Die Basis dieser Anwendung bildet die Programmiersprache Java von Sun Microsystems und wurde deshalb ausgewählt, weil sie sich im Bereich der Webanwendungen bereits etabliert hat. Einige Frameworks wie etwa Struts oder das hier verwendete JavaServer Faces bauen auf Java auf und werden für die Web-Seiten in verschiedensten Bereichen verwendet. Während der ersten Entwicklungsmonate wurde auf Version 1.5 aufgebaut.

Da aber mit der Zeit immer mehr Technologien den Weg in die Anwendung fanden und sich gewisse Abhängigkeiten bildeten wurde schliesslich auf Version 1.6 umgestellt.

2.2.2 Eclipse

Als Entwicklungsumgebung wurde das kommerzielle MyEclipse (Version 6.6) von der Firma Genuitec ausgewählt, das auf dem Open Source Framework Eclipse (Version 3.3) der Eclipse Foundation basiert und um einige Web-Tools erweitert wurde. Unter anderem bietet es eine JavaServer Faces Unterstützung (Autovervollständigung im HTML Code), die das Arbeiten mit dem Framework sehr erleichtert.

2.2.3 JavaServer Faces

JavaServer Faces, kurz JSF, ist ein Web Framework basierend auf Java. Die aktuelle Referenzimplementierung in Version 1.2 stammt von Sun Microsystems. Im Gegensatz zu anderen Frameworks, die nach dem Modell-View-Controller (MVC) Prinzip arbeiten, benutzt JSF eine komponentenbasierte Methode. Während der Benutzer eine neue Seite lädt, wird der Zustand der Komponenten in einem Komponentenbaum gespeichert und anschliessend wiederhergestellt, wenn die Seite neu geladen bzw. das Formular abgeschickt ist. Dieser Durchlauf entspricht dem JSF Lebenszyklus (Lifecycle), dabei werden sechs Phasen durchlaufen (siehe Abbildung 2.1).

Restore View (Wiederherstellung des Komponentenbaumes): In dieser Phase muss beachtet werden, ob die gewünschte Seite bereits in der Session existiert oder ob sie zum ersten Mal besucht wird. Bei der ersten Alternative wird der gespeicherte Komponentenbaum im alten Zustand wiederhergestellt, bei der zweiten Alternative wird ein neuer Komponentenbaum erstellt. Die Wiederherstellung umfasst nicht nur die Komponenten sondern auch alle damit verbundenen Validierer, Konvertierer und Backing-Beans. Letztere implementieren die Geschäftslogik für die Webseiten.

Apply Request Values (Übernahme der Anfragewerte): Einige UI-Komponenten lassen die Eingabe von Werten durch den Benutzer zu, sei es als Text oder als Auswahl von Optionen. Diese Eingaben werden durch das zugrundeliegende Formular als POST-Parameter des HTTP-Requests codiert, die Parameter vom JSF-Framework extrahiert

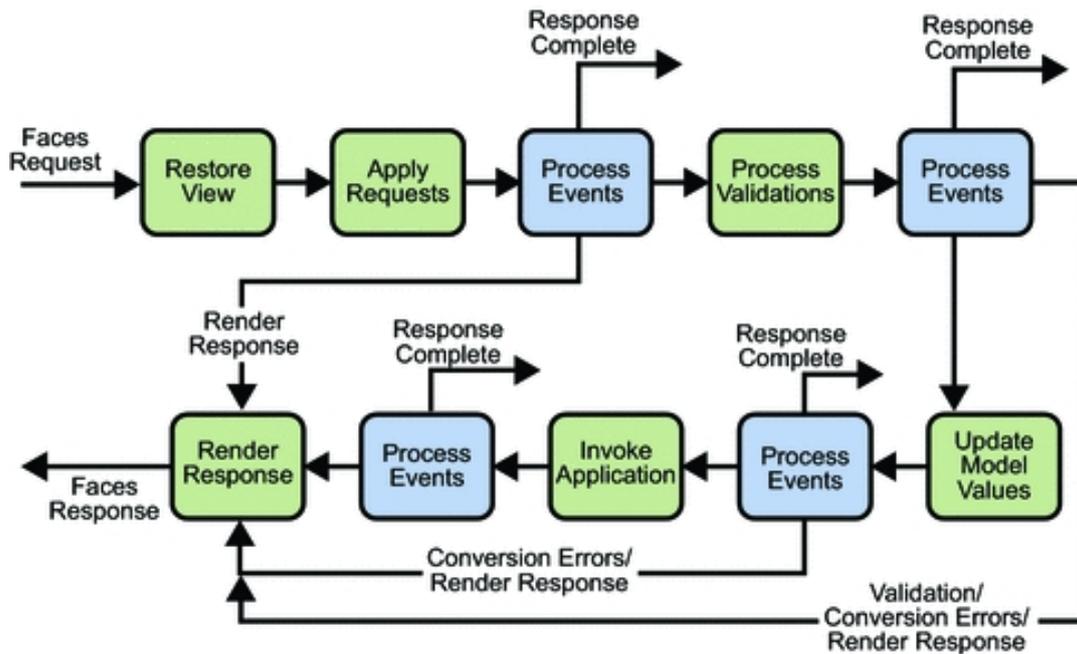


Abbildung 2.1: JSF Lifecycle [JSF]

und den entsprechenden UI-Komponenten zugewiesen.

Process Validations (Validierung): In dieser Phase wird über den gesamten Komponentenbaum iteriert und überprüft, ob alle Werte (Eingaben des Benutzers) valide sind.

Update Model Values (Aktualisierung der Modellobjekte): Bis zu diesem Zeitpunkt haben alle Vorgänge in den UI-Komponenten stattgefunden. Die Werte sind valide und vom richtigen Typ und können nun den Modellobjekten zugewiesen werden.

Invoke Application (Aufruf der Anwendungslogik): Nun kann die Anwendungslogik aufgerufen werden, d.h. es werden Aktionen wie zum Beispiel die Reaktion auf das Drücken einer Schaltfläche abgearbeitet und ausgeführt.

Render Response (Rendern der Antwort): In dieser finalen Phase wird die Antwort, zum Beispiel in Form einer XHTML Seite, erzeugt und der Komponentenbaum gespeichert. [JSF06].

Treten Fehler auf, oder soll als Antwort beispielsweise eine HTML Seite aufgerufen werden die keine JSF-Komponenten enthält, so können einzelne Phasen übersprungen werden (siehe Abbildung 2.1).

Im Rahmen dieser Anwendung wurde aber nicht die Referenzimplementierung herangezogen, sondern die Open Source Implementierung MyFaces von Apache in Version 1.2.3.

Der Grund für diese Entscheidung liegt in den zusätzlichen Komponentenbibliotheken, Tomahawk und Trinidad, die MyFaces mit sich bringt.

Tomahawk erweitert beinahe alle bestehenden Komponenten um weitere Attribute und führt zusätzlich neue ein. Der *tree* etwa bietet eine alternative Darstellung von Daten in Form eines Baumes an oder der *inputFileUpload* ermöglicht das einfache Hochladen einer Datei.

Trinidad baut ebenfalls auf den bestehenden Komponenten auf, erweitert sie in diesem Fall aber mit AJAX Funktionalität, so dass zum Beispiel ein Befehl per Knopfdruck ohne Neuladen der Seite abgesetzt werden kann. Wie bei Tomahawk führt auch Trinidad neue Komponenten, wie den Fortschrittsbalken, ein. Den Weg in die finale Version der Anwendung hat Trinidad allerdings nicht geschafft, da es nicht mit der Komponentenbibliothek Richfaces kompatibel ist, und letztere ein grösseres Spektrum an Möglichkeiten anbietet.

2.2.4 RichFaces

RichFaces wurde 2006 unter dem Namen Ajax4jsf von der Firma Exadel veröffentlicht. Zu diesem Zeitpunkt war es Teil eines Produkts namens Exadel Visual Component Plattform. Noch im selben Jahr wurde die Plattform aufgeteilt, dabei entstanden die beiden Frameworks Ajax4jsf und RichFaces. Während RichFaces einen Komponenten-basierten Ansatz verfolgte und kommerziell vertrieben wurde, bot Ajax4jsf eine seitenbezogene AJAX Unterstützung an und wurde als Open Source Projekt angeboten. 2007 vereinigten sich die beiden Firmen JBoss und Exadel, RichFaces wurde auch Open Source und schliesslich wurden Ajax4jsf und RichFaces zusammengefügt. Seitdem sind beide Produkte unter dem Namen RichFaces verfügbar.

Ein Punkt der RichFaces auszeichnet ist sein vorgefertigtes Aussehen. Der Benutzer kann mit wenigen Komponenten bereits eine sehr ansprechende Seite erstellen. Einige Style-Aspekte der Komponenten lassen sich manuell einstellen, will der Benutzer aber die generelle Struktur verändern, stösst RichFaces an seine Grenzen.

An dieser Stelle sollen noch einige wichtige Komponenten von RichFaces, die in der Applikation verwendet werden, vorgestellt werden. Die Firma JBoss unterteilt seine Komponenten je nach Funktion in Gruppen. Die folgenden Gruppen umfassen nur einen Bruchteil aller verfügbaren Komponenten, sie stellen aber die für die Anwendung essentiellen Komponenten zur Verfügung.

- **Ajax Support:** In dieser Sparte befinden sich die meisten AJAX Komponenten die benötigt werden um eine Seite AJAX tauglich gestalten zu können. Die Komponente *Poll* fordert kontinuierlich Daten vom Server an, ohne dabei die Seite neu laden zu müssen. Dazu können per Attribut Ziele angegeben werden, die nach jedem Serverzugriff aktualisiert werden. Eine weitere nützliche Komponente stellt der *Command Button* dar. Obwohl dieser bereits von MyFaces implementiert ist, erweitert RichFaces die Funktionalität um die AJAX Fähigkeit und erlaubt somit Befehle per Mausklick asynchron abzusetzen.
- **Data Iteration:** Diese Subklasse widmet sich der Datendarstellung, die zum Beispiel durch *Data List* oder *Data Table* realisiert werden kann. Obwohl MyFaces von Haus aus die meisten Komponenten dieser Kategorie liefert, bietet RichFaces mehr Funktionalität (Filter, Sortierung, Scrolling) und einen eigenen Style.
- **Rich Output:** Komponenten, die der visuell ansprechenden Darstellung dienen, befinden sich in dieser Gruppe. Die *Progress Bar* zeigt einen Fortschrittsbalken, der durch AJAX Zugriffe ständig aktualisiert wird. Das *Modal Panel* öffnet ein Popup mit gewünschtem Inhalt, das *Tab Panel* erlaubt die Unterteilung von Inhalten in mehrere Tabs und das einfache *Panel* bildet um den angegebenen Inhalt einen Rahmen mit Überschrift.
- **Rich Input:** Komponenten, die der Eingabe dienen, sind in dieser Gruppe zu finden. Eine wichtige Komponente ist der *File Upload* mit dessen Hilfe eine oder mehrere Dateien eines gewünschten Typs hochgeladen werden können. In Verbindung mit der *Progress Bar* lässt sich auch der Upload-Fortschritt mitverfolgen. Ebenfalls sehr nützlich ist die *Combo Box*, eine Mischung aus Eingabefeld und Dropdown-Liste. Der Benutzer kann hier entweder aus der Liste den gewünschten Eintrag wählen oder per Eingabefeld und Autovervollständigung (AJAX) den Eintrag finden.

2.2.5 Facelets

Eine weitere Bibliothek, die allerdings keine Komponenten zur Verfügung stellt, dafür aber eine dynamische Zusammensetzung von Seiten erlaubt, ist Facelets. Dieser alternative View-Handler ersetzt JavaServer Pages für die Definition der Views und setzt als Eingabe gültige XML-Dokumente voraus, weshalb die Seiten im XHTML-Format erstellt werden.

2.2.6 Hibernate

Bei kleineren Projekten würde es genügen, die Datenbankverbindung via JDBC manuell zu konfigurieren und anschliessend die Aufrufe (Statements) abzusetzen. Sobald jedoch der Umfang eines Projektes zunimmt und proportional dazu die Menge an Datenbankzugriffen wächst, bietet sich Hibernate an. Dieses Open Source Persistenz Framework ermöglicht es, Zustände von Objekten in einer Datenbank zu speichern ohne dabei die Zugriffe in SQL verfassen zu müssen. Wenn diese Datensätze wieder ausgelesen werden, dann erzeugt Hibernate die entsprechenden Objekte. Diese Vorgehensweise wird als Object-Relational Mapping bezeichnet. Sie erspart dem Entwickler das Verfassen von SQL Abfragen, weshalb Hibernate auch nicht auf einen bestimmten Datenbanktyp festgelegt ist. Die offizielle Definition sieht folgendermaßen aus:

"Hibernate is a powerful, high performance object/relational persistence and query service. Hibernate lets you develop persistent classes following object-oriented idiom - including association, inheritance, polymorphism, composition, and collections. Hibernate allows you to express queries in its own portable SQL extension (HQL), as well as in native SQL, or with an object-oriented Criteria and Example API." [HIB]

Die Konfiguration des initialen Verbindungsaufbaus wird in der Datei *hibernate.cfg.xml* festgelegt. Darin lassen sich zahlreiche Einstellungen vornehmen, die wichtigsten werden an dieser Stelle aufgelistet:

```
1 <hibernate-configuration>
2   <session-factory>
3     <!-- Database connection settings -->
4     <property name="connection.driver_class">com.mysql.jdbc.Driver</property>
5     <property name="connection.url">jdbc:mysql://localhost:3310/education</property>
6     <property name="connection.username">root</property>
7     <property name="connection.password">test</property>
8
9     <!-- SQL dialect -->
10    <property name="dialect">org.hibernate.dialect.MySQLDialect</property>
11
12    <!-- Drop and re-create the database schema on startup -->
13    <property name="hbm2ddl.auto">update</property>
14
15    <!-- Mapping of resource files -->
16    <mapping resource="beans/UserBean.hbm.xml"/>
```

```

17 <mapping resource="beans/AnnouncementBean.hbm.xml"/>
18 <mapping resource="beans/QuestionBean.hbm.xml"/>
19 <mapping resource="beans/QuizBean.hbm.xml"/>
20 <mapping resource="beans/AnswerBean.hbm.xml"/>
21 <mapping resource="beans/AssignmentBean.hbm.xml"/>
22 <mapping resource="beans/UsergroupBean.hbm.xml"/>
23 </session-factory>
24 </hibernate-configuration>

```

Listing 2.1: hibernate.cfg.xml

Zuerst werden alle Daten (Treiber, URL, Username, Password) für den Verbindungsaufbau mit der Datenbank angegeben. Als nächstes muss der SQL Dialekt festgelegt werden, damit Hibernate weiss, welche Datenbank verwendet wird. Die Eigenschaft *hbm2ddl.auto* bestimmt, wie das Erstellen der Tabellen abläuft, *update* erzeugt nur Tabellen die noch nicht existieren bzw. aktualisiert bereits bestehende, *create* hingegen löscht zuerst alle Tabelle und erzeugt sie anschliessend wieder neu. Schliesslich werden noch alle XML Mapping Dateien aufgelistet. Listing 2.2 verdeutlicht den Aufbau einer Mapping Datei anhand eines Beispiels.

```

1 <hibernate-mapping>
2 <class name="beans.UserBean" table="USERBEAN" >
3 <id name="id" column="USER_ID" not-null="true"/>
4 <property name="name" column="USER_NAME" not-null="true"/>
5 <property name="password" column="USER_PASSWORD"/>
6 <property name="role" column="USER_ROLE" not-null="true"/>
7 </class>
8 </hibernate-mapping>

```

Listing 2.2: UserBean.hbm.xml

Diese Hibernate Mapping Datei dient dem Zweck, aus Java Klassen persistente Objekte zu erzeugen. Listing 2.2 beschreibt das Mapping der Klasse `UserBean` (`class name="beans.UserBean"`) auf die Tabelle `USERBEAN` (`table="USERBEAN"`). Für alle zu persistierenden Variablen in der Klasse werden über Properties (`<property name="name" column="USER_NAME"/>`) die Spalten in der Tabelle angelegt bzw. verknüpft.

Alternativ zu dem Mapping via XML Dateien können die Verknüpfungen zwischen Java-Objekten und Datenbank-Tabellen über Java Annotations (ab Java 1.5) hergestellt werden. Im Rahmen dieser Arbeit wurde aber darauf verzichtet, da sich kleinere Projekte

sehr gut und schnell mit Mapping Dateien realisieren lassen.

Wie in der Einleitung zu diesem Abschnitt erwähnt wurde, operiert Hibernate auf Objekten. Jedes Objekt kann einen von drei Zuständen einnehmen: *Persistent*, *Detached* und *Transient*. In Abbildung 2.2 kann man die Übergänge zwischen den drei Zuständen betrachten.

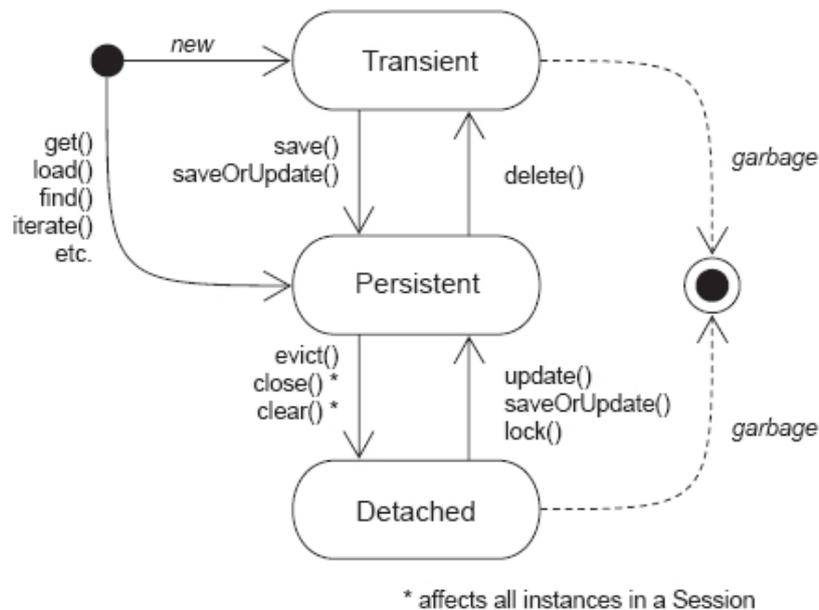


Abbildung 2.2: Hibernate Lifecycle [HIBPER]

Wenn ein neues Objekt erzeugt wird, befindet es sich im Zustand *Transient*, das heißt das Objekt ist nur im Speicher vorhanden und noch mit keiner Datenbank Tabelle verknüpft. Objekte in diesem Zustand werden nicht als Transaktions-Objekte zugelassen, weshalb alle Änderungen darauf verloren gehen. Um in den Zustand *Persistent* zu gelangen muss entweder ein bereits persistiertes Objekt aus der Datenbank geladen werden (*get*, *load*) oder ein neu erzeugtes nach Hibernate Konvention gespeichert werden (*save*, *saveOrUpdate*). Im Gegensatz zum Zustand *Transient* werden in *Persistent* nach dem Commit alle Änderungen übernommen und in die Datenbank gespeichert (*Dirty Checking*). Objekte, die sich im Zustand *Detached* befinden, werden nicht länger mit der Datenbank synchronisiert. Zwar existieren sie nach wie vor in der Datenbank und im Speicher, jedoch werden alle folgenden Änderungen nicht mehr persistiert. In diesen Zustand gelangt man, nachdem zum Beispiel die Session geschlossen wurde. Ein *detached*

Objekt kann aber jederzeit gespeichert werden (update, saveOrUpdate), vorausgesetzt es existiert eine offene Session. Alle Objekte, die sich im Zustand *Transient* oder *Detached* befinden und dereferenziert werden, fallen dem Garbage Collector zum Opfer.

2.2.7 Datenbank: MySQL oder HSQLDB

Hinter den meisten modernen Web-Anwendungen steht heutzutage eine Datenbank, sei es zum Verwalten der Benutzer oder zum Speichern ganzer Inhalte. Es gibt eine Vielzahl an kommerziell und frei verfügbaren Datenbanken, jede hat ihre Vor- und Nachteile. Im Rahmen der Entwicklung dieser Anwendung wurde anfangs als Datenbank HSQLDB ausgewählt, eine vollständig in Java programmierte relationale SQL-Datenbank, die unter einer eigenen Open Source Lizenz steht. HSQLDB zeichnet sich durch seine geringe Grösse auf der Festplatte (< 1 Megabyte Speicherbedarf) und durch seine Geschwindigkeit aus, und ist deshalb für Testzwecke sehr gut geeignet. Nichtsdestotrotz fehlen einige essentielle Elemente, die eine vollwertige Datenbank auszeichnen, weshalb während der Implementierung der Wechsel auf MySQL Server erfolgte. MySQL Server ist ein Relationales Datenbankverwaltungssystem und als Open Source Software für verschiedene Betriebssysteme verfügbar. Das Produkt wurde von der schwedischen Firma MySQL AB entwickelt, jedoch im Februar 2008 von der Firma Sun Microsystems übernommen, die nun für die Weiterentwicklung des Codes verantwortlich ist. Neben der weiterhin frei verfügbaren Version, die unter der General Public License (GPL) steht, wird auch eine kommerzielle Version angeboten.

MySQL Server wurde für die Anwendung ausgewählt, da es frei verfügbar ist und es sich dank Hibernate (siehe Abschnitt 2.2.6) problemlos in das bestehende System integrieren lässt. Es kann aber auch jederzeit eine andere Datenbank verwendet werden.

2.2.8 Ant

Das Open Source Projekt Ant von Apache wird für das automatische Bauen der Anwendung herangezogen und erleichtert die Projektgenerierung erheblich. Ant ist in MyEclipse integriert und bedarf somit keiner weiteren Installation. Für das Bauen des Projekts ist lediglich eine Datei namens build.xml Voraussetzung, ein Beispiel dafür ist in Listing 2.3 ersichtlich. Damit erfüllt Ant dieselbe Aufgabe wie das Programm make, das ein

Makefile abarbeitet.

```
1 <project name="diplom" default="war">
2
3 <property name="build.directory" value="WebRoot/WEB-INF/classes"/>
4 <property name="source.java.directory" value="src"/>
5 <property name="libraries.directory" value="WebRoot/WEB-INF/lib"/>
6 <property name="war.filename" value="${ant.project.name}.war"/>
7
8 <path id="libraries">
9 <fileset dir="${libraries.directory}">
10 <include name="*.jar" />
11 </fileset>
12 </path>
13
14 <target name="clean" unless="clean.performed">
15 <delete file="${war.filename}"/>
16 <delete dir="${build.directory}"/>
17 <property name="clean.performed" value="true" />
18 </target>
19
20 <target name="compile" depends="clean, copy.resources">
21 <mkdir dir="${build.directory}"/>
22 <javac srcdir="${source.java.directory}" destdir="${build.directory}"
23 debug="on" source="1.5" nowarn="false" failonerror="true">
24 <classpath refid="libraries"/>
25 </javac>
26 </target>
27
28 <target name="copy.resources">
29 <copy todir="${build.directory}">
30 <fileset dir="${source.java.directory}">
31 <exclude name="**/*.java"/>
32 </fileset>
33 </copy>
34 </target>
35
36 <target name="war" depends="compile">
37 <war destfile="\${war.filename}" webxml="WebRoot/WEB-INF/web.xml">
38 <fileset dir="WebRoot"/>
39 </war>
40 </target>
41
```

```

42 <target name="run" depends="compile">
43   <java fork="true" classname="services.HibernateManager" classpathref="libraries">
44     <classpath path="\${build.directory}"/>
45     <arg value="action" line="list"/>
46   </java>
47 </target>
48
49 </project>

```

Listing 2.3: build.xml

Im ersten Teil werden einige Werte definiert, wie zum Beispiel das Verzeichnis für die kompilierten Klassen, den Source-Ordner, den Library-Ordner oder der Name der finalen WAR Datei. Anschliessend werden mehrere sogenannte *Targets* definiert, dabei handelt es sich um ausführbare Befehle, die auch untereinander Abhängigkeiten aufweisen können. Während der Entwicklung wird in der Regel der Befehl *compile* gewählt, der zuerst ein *clean* und *copy.resources* ausführt bevor die Quelldateien kompiliert werden. Schliesslich erzeugt das *Target war* eine WAR Datei, die mit Hilfe von Tomcat geladen und der Inhalt in einem Webbrowser getrachtet werden kann.

2.2.9 Tomcat

Tomcat ist ein weiteres Open Source Projekt von Apache. Der in Java geschriebenen Servlet-Container wird für die Ausführung von Java Code auf Webservern verwendet. Während der Entwicklung hat auch in diesem Bereich ein Wechsel von Version 5 auf 6 stattgefunden, da Tomcat 6 Voraussetzung für die aktuelle MyFaces Implementierung ist.

2.2.10 XML

Wie die bisherigen Listings zeigen, basieren beinahe alle Konfigurationsdateien auf XML. Die Anwendung beschränkt sich aber nicht nur darauf, sondern integriert XML auch bei der Reportgenerierung. Nachdem eine Umfrage im QTI Format durchgeführt wurde, liefert die Applikation eine Zusammenfassung in XML Form. Mit Hilfe der Programmiersprache XSLT (eXtensible Stylesheet Language Transformation), die von der Java

Bibliothek unterstützt wird, können diese Reports anschliessend in HTML Code umgewandelt und dem Benutzer im Browser angezeigt werden.

2.2.11 Log4j

Fehler lassen sich während der Entwicklung nicht vermeiden und müssen deshalb ausfindig gemacht und beseitigt werden. Aber auch nach dem Abschluss der Programmierung können im Live Betrieb noch Fehler auftreten. In manchen Situationen ist es allerdings nicht immer auf den ersten Blick ersichtlich wo die Ursache liegt. Man kann sich die Arbeit erheblich erleichtern indem man auf das Logging zurückgreift, das heisst es wird an kritischen Stellen im Quellcode eine Ausgabe in eine Logdatei erzwungen. Obwohl Java bereits ein Logging Konzept mitliefert, bietet Log4J von Apache ein flexibleres System mit mehr Auswahlmöglichkeiten. Die Konfiguration für das Logging wird in der Datei `log4j.properties` vorgenommen.

2.2.12 Zusammenfassung

In Tabelle 2.1 werden nochmals alle zuvor besprochenen Technologien aufgelistet und die benutzte Version und der Verwendungszweck angegeben.

Technologie	Version	Verwendungszweck
Java	1.6.1	Programmiersprache für die Web-Anwendung
MyEclipse	6.6	Entwicklungsumgebung
JavaServer Faces (MyFaces)	1.2.3	Framework-Standard für Webapplikationen
Tomahawk	1.1.3	Komponentenbibliothek
RichFaces	3.2.2	Komponentenbibliothek
MySQL Server	5.0	Datenbank
Hibernate	3.2.5	Persistenz-Framework
Ant	1.7.0	Build-Werkzeug
Tomcat	6.0	Web-Container
Log4J	1.2.11	Logging-Framework
QTI		QTI Framework

Tabelle 2.1: Überblick über die verwendeten Technologien

2.3 QTI

In diesem Unterkapitel soll der QTI-Aspekt in der Anwendung genauer beleuchtet werden. Dazu wird zuerst das verwendete Framework mit seinen Vor- und Nachteilen beschrieben. Anschliessend wird die Kommunikation zwischen Anwendung und Framework im Detail erklärt, beginnend beim Verbindungsaufbau über den Datentransfer bis hin zur Reportgenerierung. Da das QTI-Framework nicht für die Benutzung innerhalb des JSF Frameworks ausgelegt war, mussten einige Anpassungen durchgeführt werden. Im letzten Abschnitt werden die benötigten Änderungen erläutert.

2.3.1 QTITools

Obwohl eine Vielzahl an Webanwendungen existieren, die QTI unterstützen, ist es nicht einfach ein modulares Open Source QTI-Framework in Java zu finden, das man für seine eigenen Zwecke anpassen und verwenden kann. Die meisten Anwendungen verwenden eine eigens entwickelte QTI Umgebung, die sich nicht so ohne Weiteres für andere Applikation verwenden lässt. Entweder sie sind nur kommerziell verfügbar, nicht in Java geschrieben oder derart programmiert, dass sie nur durch einen hohen Mehraufwand und Adaptionen benutzbar wären.

Die University of Southampton in England hat 2008 das Projekt QTITools, das Teil des ASDEL (Assessment Delivery Engine) Projekts an der School of Electronics and Computer Science ist, abgeschlossen. Dabei handelt es sich um eine Ansammlung von Frameworks, die dem Erzeugen, Durchführen und Speichern von QTI Umfragen dienen. Der komplette Quellcode ist in Java geschrieben und alle Frameworks sind als Open Source Projekte frei verfügbar. Die Basis für diese Frameworks bildet die eigens entwickelte Bibliothek JQTI. Mit Hilfe dieser Bibliothek können IMS QTI v2.1 Dateien im XML Format gelesen, übersetzt und interpretiert werden. Auch das Erstellen von QTI Umfragen kann mit Hilfe von JQTI erreicht werden. Weiters führt JQTI eine umfassende Validierung durch und generiert im Fehlerfall Meldungen.

In der folgenden Auflistung werden die einzelnen QTITools Projekte im Detail betrachtet:

- **constructr:** Mit Hilfe dieses Frameworks lassen sich neue QTI Umfragen erstellen. Der *constructr* bietet dem Benutzer die Möglichkeit, die Fragen für die Umfrage aus

einer *item-bank* auszuwählen und zu einem Content Package zusammenzusetzen, das anschliessend vom *playr* und *assessr* geladen werden kann.

- **playr:** Bereits erstellte Content Packages können im *playr* geladen und ausgeführt werden.
- **assessr:** Während der *playr* die Fragen visualisiert, zeichnet der *assessr* die Ergebnisse jeder abgeschlossenen Frage auf und liefert zum Schluss eine Bewertung bzw. Zusammenfassung.
- **validatr:** Bevor Umfragen veröffentlicht werden, können sie mit dem *validatr* überprüft werden. Er markiert alle Fehlerstellen und warnt darüber hinaus vor möglichen Problemen. Sind alle Fehler bereinigt, lässt sich im *validatr* die Abfolge der Fragen im Content Package graphisch darstellen.
- **R2Q2:** Hinter diesem kryptischen Namen steckt eine *rendering and responding engine* für QTI v2.0 Fragen. Während der *playr* die Fragen darstellt, übernimmt R2Q2 die dahinterliegende Logik, d.h. alle vom Benutzer gemachten Eingaben werden an R2Q2 weitergegeben und dort bearbeitet. Danach wird der Code der zu rendernden Seite von R2Q2 an den *playr* zurückgeliefert und angezeigt. Die detaillierte Funktionsweise von R2Q2 folgt im nächsten Abschnitt.

Im Rahmen dieser Arbeit sind nur die beiden Frameworks *playr* und *R2Q2* von Bedeutung. Das Erzeugen und Validieren von eigenen Content Packages kann im Zuge von Erweiterungen noch hinzugefügt werden, für die aktuelle Version der Anwendung ist das allerdings nicht vorgesehen.

2.3.2 Die Kommunikation

Wie kann man den *playr* nun in eine fremde Anwendung integrieren? In weiser Voraussicht haben die Programmierer der QTITools ihre Frameworks so implementiert, dass sie auch über Webservices angesprochen werden können. In Abbildung 2.3 wird der interne Aufbau des *playr* dargestellt.

Wie aus der Grafik hervorgeht, besteht der *playr* aus drei Teilen, der *TestControllerEngine*, der *QTIAssessmentTestEngine* und der *AssemblerRendererEngine*. Die *TestControllerEngine* ist für die Kommunikation nach aussen verantwortlich. Sobald den *playr* über das Webservice Interface eine Anfrage erreicht, wird eine neue Session ID vom Ses-

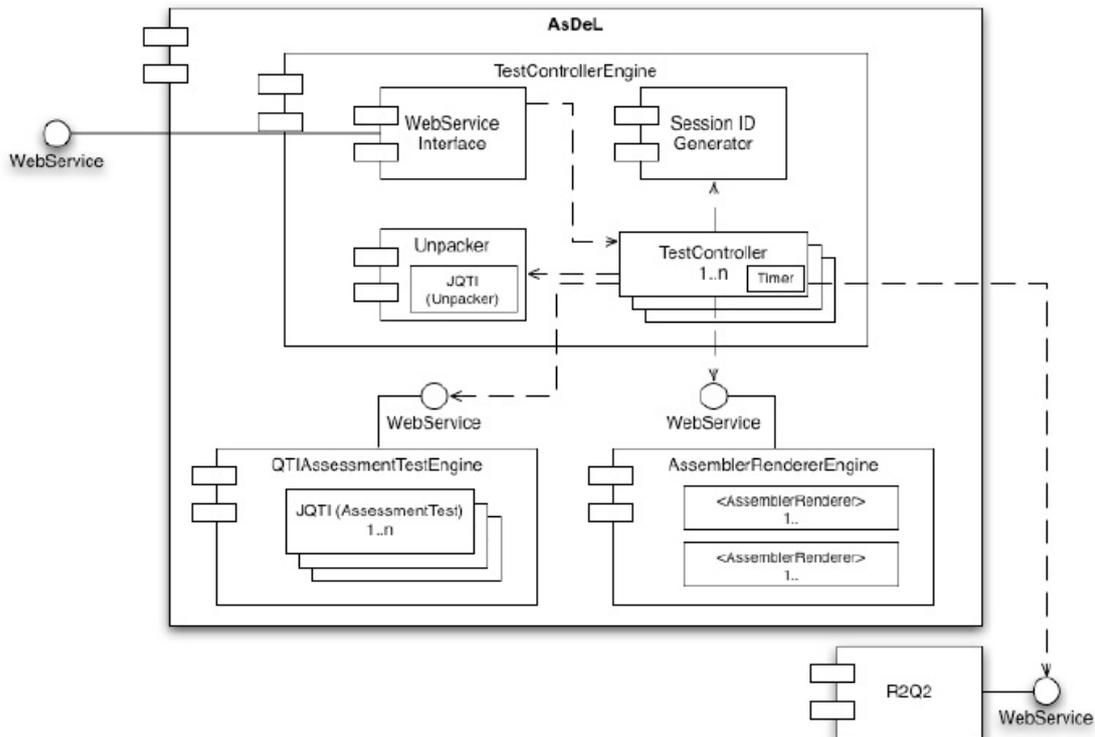


Abbildung 2.3: ASDEL [ASDEL1]

session ID Generator erzeugt, die während dem gesamten Datenaustausch gültig ist und den Benutzer identifiziert. Eine weitere Aufgabe der *TestControllerEngine* liegt im Entpacken und Validieren des eintreffenden Content Package durch den Unpacker. Ist bis zu diesem Zeitpunkt kein Fehler aufgetreten, dann erhält der *TestController* von der *QTIAssessmentTestEngine* den Auftrag, die erste Frage zu bearbeiten. Dazu übermittelt der *TestController* die erste Frage an das externe Framework *R2Q2*, das anschliessend die Frage bearbeitet. Nachdem die Frage ausgewertet wurde, wird sie vom *TestController* weiter an die *AssemblerRendererEngine* weitergereicht und dort in HTML Code eingebettet. Schliesslich wird die fertig gerenderte Seite an den Benutzer zurückgegeben und kann von ihm beantwortet werden. Dieses Prozedere läuft so lange bis die *QTIAssessmentTestEngine* meldet, dass die letzte Frage abgehandelt wurde.

Bis jetzt kennt man das Framework *R2Q2* nur als Black Box, eine Frage wird übergeben, ausgewertet und schliesslich wieder zurückgeliefert. Wie die genaue Funktionsweise aussieht, zeigt Abbildung 2.4.

R2Q2 besteht aus vier Elementen: *Router*, *Initialiser*, *Renderer*, *Processor*. Der *Router* übernimmt die Koordination zwischen den einzelnen Operatoren. Zuerst zerlegt er

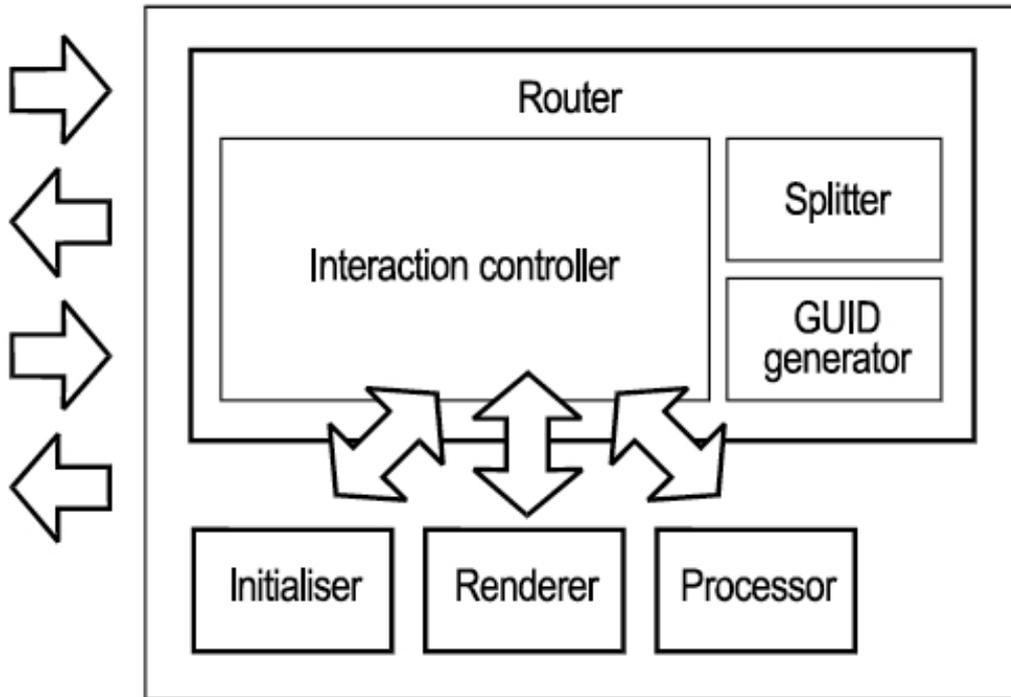


Abbildung 2.4: R2Q2 [ASDEL2]

das eintreffende *Item* (Frage) mit einem *Splitter*, anschliessend wird allen daraus resultierenden Fragmenten eine ID zugewiesen und das *Item* an den *Initialiser* weitergereicht. Dieser initialisiert das *Item*, indem er alle benötigten Ausgangs- und Antwortvariablen generiert. Der *Processor* verarbeitet im Anschluss die Antwort des Benutzers (falls vorhanden), indem er einen Satz von Regeln anwendet und generiert auf Basis deren eine Antwort. Schliesslich rendert der *Renderer* die vom *Processor* erhaltene Antwort durch Umwandeln des XML in HTML Code und der Router liefert das Resultat an den Benutzer zurück.

Nachdem nun der interne Aufbau des playr- und R2Q2-Frameworks anschaulich dargestellt wurde, fehlt noch die Anbindung an die eigene Anwendung. Abbildung 2.5 zeigt, wie die Kommunikation zwischen Anwendung und playr abläuft.

Anmerkung:

Es werden alle Befehle, die an den playr adressiert sind, an

`http://playr_server:playr_port/path_to_playr/`

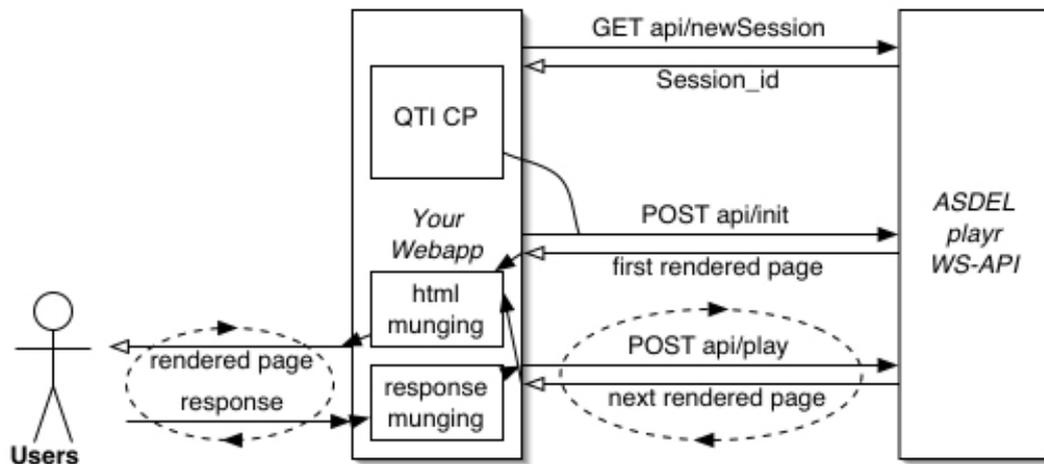


Abbildung 2.5: Kommunikationsablauf [QTITOOLS]

geschickt. Um den Rahmen nicht zu sprengen werden in der folgenden Erklärung alle Pfade relativ zu diesem gesehen angegeben.

Im ersten Schritt muss eine neue Session geöffnet werden. Dazu muss nur ein GET Request an `/api/newSession` gesendet werden, die neue Session ID befindet sich in der Antwort und muss sofort gespeichert werden, da alle weiteren Operationen darauf zurückgreifen. Anschliessend kann die Session initialisiert werden indem ein Multipart POST Request an `/api/init;jsessionid=the_session_id` gesandt wird. In diesem Request muss unter anderem das zu ladende Content Package mitgegeben werden. Als Antwort darauf erhält man die erste gerenderte Seite, die nur mehr in der Anwendung angezeigt werden muss. Nachdem der Benutzer die erste Frage beantwortet und abgeschickt hat, wird seine Eingabe in Form von Parametern in einem POST Request an `/api/play;jsessionid=the_session_id` weitergeleitet. Dieses Prozedere wiederholt sich, bis alle Fragen abgeschlossen sind, wobei jederzeit der aktuelle Status mit einem GET Request an `/api/report;jsessionid=the_session_id` abgefragt werden kann, vorausgesetzt die Session ist noch offen. Ob die Umfrage zu Ende ist, kann über einen GET Request an `/api/introspect/isCompleted;jsessionid=the_session_id` abgefragt werden.

2.3.3 Anpassungen

Die Entwickler der QTITools-Frameworks haben ein Wiki [QTITOOLS] in das Internet gestellt, um Benutzern den Umgang mit ihren Frameworks zu erleichtern. Leider waren die darin beschriebenen Beispiele nicht direkt auf die JSF-Umgebung übertragbar, weshalb einige Adaptionen durchgeführt werden mussten. Einerseits ist es wichtig, die vom Benutzer gegebenen Antworten in Form von Parametern korrekt an den *playr* zu übergeben, andererseits müssen die eingebundenen JavaScript Dateien überarbeitet werden, damit alle Typen von QTI Fragen fehlerfrei angezeigt werden. Auf die genannten Anpassungen wird im Abschnitt 2.7.3 hingewiesen.

2.4 Die Implementierung

In diesem Unterkapitel wird die Anwendung aus der technischen Sicht betrachtet. Zuerst wird die Verzeichnisstruktur dargestellt, anschliessend der Aufbau der Ansicht mit Hilfe von *Facelets* besprochen. Einen grossen Teil dieses Abschnitts nimmt die Beschreibung der Webseiten und benutzen Komponenten in Kombination mit der dahinterstehenden Geschäftslogik ein. Dazu gehört auch die Datenbankanbindung mittels Hibernate. Weitere interessante Punkte sind das Rollen-System, Logging, Filter, Properties und Styles.

2.4.1 Struktur der Anwendung

JavaServer Faces setzt, wie die meisten Web-Frameworks, eine gewisse Verzeichnisstruktur voraus. Drei Kriterien müssen erfüllt sein um ein JSF Projekt erfolgreich starten zu können. Erstens müssen alle benötigten Bibliotheken im Verzeichnis WEB-INF/lib vorhanden sein. Zweitens muss die Datei web.xml mit der minimalen Konfiguration existieren und drittens wird die Datei faces-config.xml im Ordner WEB-INF benötigt. In Abbildung 2.6 wird die komplette Verzeichnisstruktur der Applikation dargestellt.

Neben den oben genannten Kriterien wurde die Struktur noch um einige Ordner und Dateien erweitert.

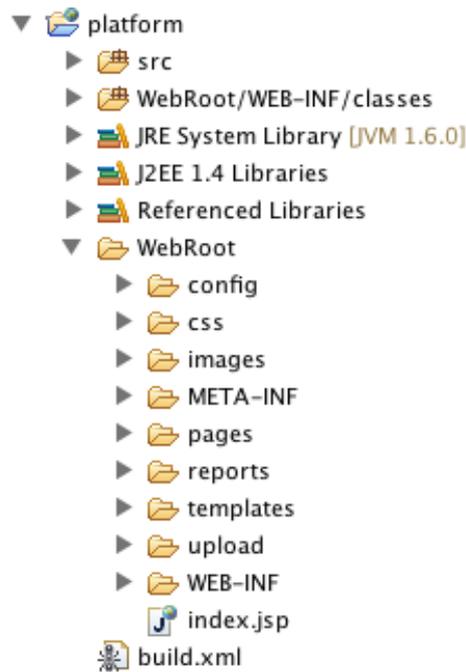


Abbildung 2.6: Verzeichnisstruktur

- *src* beinhaltet alle Java Quellcode-Dateien, die jedoch in der veröffentlichten Version nicht enthalten sind.
- *WebRoot/config* stellt die Datei `configuration.properties`, über die verschiedene Pfade gesetzt werden können, zur Verfügung.
- Im Ordner *WebRoot/css* befindet sich der Stylesheet `styles.css`, in dem das Aussehen der HTML Komponenten definiert werden kann.
- *WebRoot/images* beinhaltet alle in den Webseiten eingebundenen Bilder.
- Alle angezeigten Webseiten befinden sich im Ordner *WebRoot/pages*.
- Abgeschlossene QTI Umfragen werden im XML und HTML Format im Ordner *WebRoot/reports* gespeichert.
- Im Ordner *WebRoot/templates* liegen die XSL Dateien für die Report-Transformation.
- Hochgeladene Content Packages werden in den Ordner *WebRoot/upload* gespeichert.
- Alle kompilierten Java Klassen, Hibernate Mapping-Dateien, die Hibernate Konfiguration (`hibernate.cfg.xml`) und die Log4j Einstellungen (`log4j.properties`) werden in den Ordner *WebRoot/WEB-INF/classes* kopiert.

Eine detailliertere Auflistung aller Klassen und Webseiten befindet sich in Abschnitt 2.4.3 und 2.4.4.

2.4.2 Aufbau der Ansicht mit Facelets

Wie bereits in Abschnitt 2.2.3 erwähnt, wird für die Anzeige der Seiten nicht JSP als Technologie gewählt, sondern Facelets. Der Vorteil liegt in der dynamischen Zusammensetzung der Seiten. Mussten vorher alle Seiten von Grund auf neu gestaltet werden, erlaubt Facelets jetzt das Erzeugen eines Templates, auf dem alle folgenden Seiten basieren. In diesem Template könnte zum Beispiel ein Header definiert werden, der in allen Seiten integriert wird, ohne den Code wiederholt schreiben zu müssen. Im konkreten Fall dieser Arbeit werden im Template die folgenden vier Teilbereiche definiert:

1. Header: Im Kopf, auch Header genannt, befindet sich die Überschrift der Anwendung, der Name des angemeldeten Benutzers und der Logout Knopf.
2. Sidebar: Die Sidebar dient der Navigation innerhalb der Anwendung.
3. Footer: Die Fusszeile zeigt in den meisten Fällen keinen Inhalt an. Sie dient mehr als vertikaler Abgrenzer und kann noch durch Information erweitert werden (z.B. Impressum, Kontakt etc.).
4. Body: Der Body stellt den Hauptbereich dar, der dem Benutzer angezeigt wird. Während Header, Footer und Sidebar stets unverändert bleiben, wird der Body dynamisch befüllt.

In Listing 2.4 wird zuerst das Template präsentiert, anschliessend zeigt Listing 2.5, wie das Template in einer Seite (example.xhtml) verwendet wird.

```
1 <html >
2   <head >
3     <ui:insert name="head">
4       <title>Education Platform</title>
5       <link rel="stylesheet" type="text/css" href="../../css/styles.css" />
6       <ui:insert name="javascript" >
7         <script language="JavaScript">
8           function startup() {
9             }
10        </script>
11      </ui:insert >
12    </ui:insert >
13  </head >
```

```

14
15 <body>
16   <ui:include src="header.xhtml"/>
17
18   <table class="templatetable">
19     <tr>
20       <td class="sidebarempty">
21         <ui:include src="sidebar.xhtml"/>
22       </td>
23       <td class="body">
24         <span class="headline">
25           <ui:insert name="headline"/>
26         </span>
27         <ui:insert name="body" />
28         <rich:spacer width="1" height="5"/>
29       </td>
30     </tr>
31   </table>
32
33   <ui:include src="footer.xhtml" />
34 </body>
35 </html>

```

Listing 2.4: template.xhtml

```

1 <html>
2   <ui:composition template="../includes/template.xhtml">
3     <head>
4       <title>My Facelets Page</title>
5       <!--<link rel="stylesheet" type="text/css" href="styles.css"-->
6     </head>
7     <body>
8       <ui:define name="body">
9         ...
10      </ui:define>
11    </body>
12  </ui:composition>
13 </html>

```

Listing 2.5: example.xhtml

Wie in den beiden Listings zu erkennen ist, gibt es einige essentielle Tags:

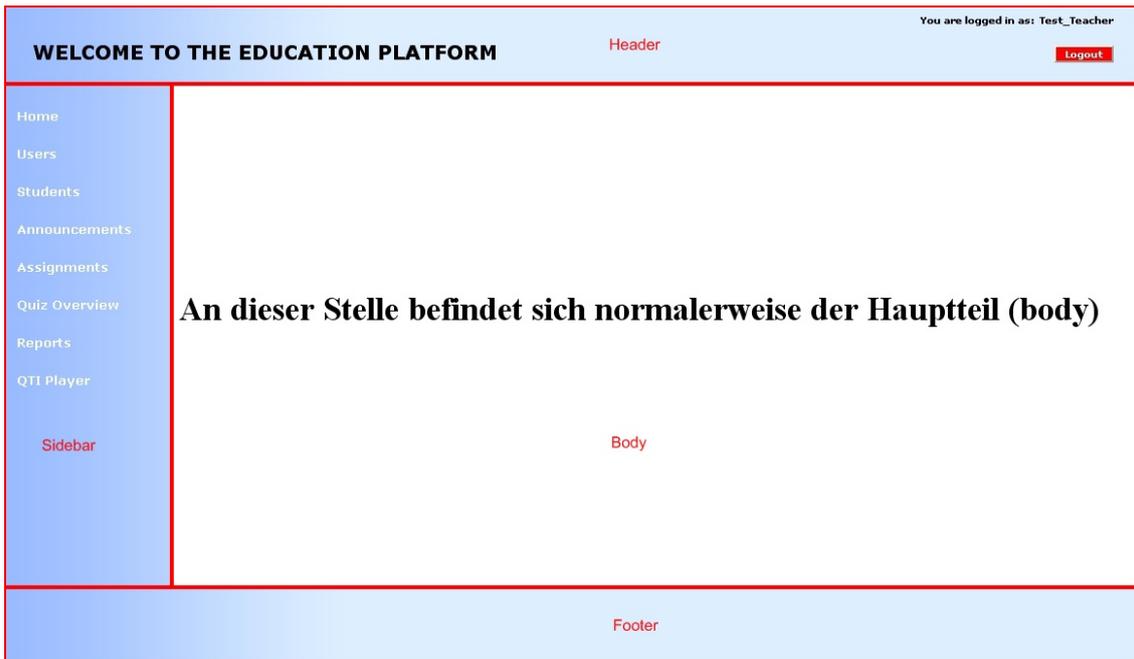


Abbildung 2.7: Seite mit leerem Hauptteil

- `<ui:include>` bindet eine andere bestehende Webseite an der gewünschten Stelle ein
- `<ui:insert>` markiert im Template die Stellen, die anschliessend in der Webseite ersetzt werden können
- `<ui:define>` wird in einer Webseite verwendet um den in das Template zu integrierenden Bereich zu markieren
- `<ui:composition>` wird verwendet um auf einer Webseite das zu benutzende Template zu definieren

Eine zwingende Voraussetzung bringt Facelets allerdings mit sich, es müssen alle Webseiten im xhtml Format vorliegen.

2.4.3 Webseiten, integrierte Komponenten und AJAX

Der visuelle Teil der Anwendung umfasst insgesamt 29 Webseiten, die sich in mehrere Subkategorien untergliedern.

Kategorie	Webseiten
Nachrichten	announcementOverview, createAnnouncement
Aufgaben	assignmentOverview, createAssignment, preview, showAssignment, solveAssignment
Facelets	footer, header, sidebar, template
Anmeldung	login, visitor
Workspace	myWorkspace
QTI	qtiPlayer
Umfragen	chooseQti, createQuiz, quiz, quizDialogue, quizOverview, report, setAnswers, showQuiz, takeQuiz
Resultate	reportsOverview
Studenten	studentOverview
Benutzer	createUser, userOverview

Tabelle 2.2: Webseiten-Kategorien

Alle in Tabelle 2.2 aufgelisteten Kategorien werden in den folgenden Abschnitten im Detail beschrieben.

Nachrichten

Die Seite `announcementOverview.xhtml` dient dem Teacher als Übersicht, welche Nachrichten er bereits für die Studenten verfasst hat. Er kann Nachrichten editieren, löschen oder neue verfassen. Will der Teacher eine neue Nachricht schreiben, wird er automatisch auf die Seite `createAnnouncement.xhtml` weitergeleitet.

Aufgaben

Ein weiteres Feature des Teachers ist das Anlegen von Aufgaben (`assignment`). Auf der Seite `assignmentOverview.xhtml` werden alle bisher erstellten Aufgaben aufgelistet und Bearbeitungsmöglichkeiten wie Löschen und Zuweisen einer Gruppe angeboten. Neue Aufgaben werden auf der Seite `createAssignment.xhtml` erstellt und können anschließend auf der Seite `preview.xhtml` als Vorschau betrachtet werden. Für den Student existiert

tieren zwei Seiten, `showAssignment.xhtml` und `solveAssignment.xhtml`. Wie die Namen bereits verraten, kann auf ersterer Seite eine Aufgabe betrachtet und auf letzterer beantwortet werden.

Facelets

Alle Dateien in diesem Ordner sind nicht dafür konzipiert, alleine angezeigt zu werden. Vielmehr handelt es sich um Teile einer Webseite, die in andere Webseiten eingebunden werden. Wie bereits im Kapitel 2.4.2 angesprochen, werden für das dynamische Zusammenstellen von Webseiten mit Facelets eine Vorlage (`template.xhtml`) und einzubindende Elemente (`footer.xhtml`, `header.xhtml`, `sidebar.xhtml`) benötigt.

Anmeldung

Diese Kategorie beinhaltet zwei Webseiten: `login.xhtml` ist für den gewöhnlichen Anmeldevorgang verantwortlich während `visitor.xhtml` dieses Prozedere überspringt und sofort die Startseite anzeigt. Wählt man letzteren Einstieg, ist man anonym am System angemeldet und sehr eingeschränkt was die Interaktionsmöglichkeiten anbelangt. Der Weg über den `visitor` wird nur für die Durchführung von anonymen Umfragen gewählt.

Workspace

In diesem Ordner befindet sich nur die gleichnamige Seite `myWorkspace.xhtml`, die als Startseite nach dem Login fungiert, egal ob man als Teacher oder Student angemeldet ist.

QTI

Die Seite `qtiPlayer.xhtml` ist nur für den Teacher angedacht, damit er vorab ein QTI Content Package betrachten kann, bevor er es den Studenten als Umfrage zur Verfügung stellt.

Umfragen

Dieser Ordner bildet das Kernstück der Anwendung, er enthält alle Seiten, die für die Erstellung und Durchführung einer Umfrage benötigt werden. Die Seite `chooseQti.xhtml` ermöglicht es dem Teacher, eine Umfrage auf Basis eines QTI Content Packages zu erstellen. Die Alternative besteht darin, Fragen manuell zu verfassen und zu einer Umfrage zusammenzusetzen. Diese Methode wird mittels der beiden Seiten `createQuiz.xhtml` und `setAnswers.xhtml` realisiert. Sie erlaubt dem Teacher, Umfragen mit folgenden 6 Fragetypen zu erstellen:

- Eine korrekte Antwort - "Radiobuttons"
- Mehrere korrekte Antworten - "Checkboxes"
- Auswahl aus einer Drowdown-Box
- Bewerten in Form eines Ratings, d.h. der Teilnehmer vergibt eine Zahl zwischen 1 (Sehr Gut, Zutreffend) und 5 (Nicht Genügend, Nicht zutreffend)
- Freie Eingabe in einer Zeile
- Freie Eingabe mit mehreren Zeilen

Generierte Umfragen werden anschliessend auf der Seite `quizOverview.xhtml` aufgelistet und können hier vom Teacher auch gelöscht oder ausgeführt werden. Wählt er letztere Option, wird die Seite `showQuiz.xhtml` geladen. Sie bietet dem Teacher die Möglichkeit die geöffnete Umfrage zu starten oder abubrechen, Gruppen zuzuweisen und den Fortschritt während der Durchführung zu verfolgen. Ist eine Umfrage zu Ende, dann können alle Resultate auf der Seite `report.xhtml` betrachtet werden.

Die am System angemeldeten Teilnehmer betreten die Seite `takeQuiz.xhtml` um zu erfahren, ob bereits eine Umfrage offen ist. Wenn ja, können sich alle Benutzer, die den Umfrage-Anforderungen (richtige Gruppe) entsprechen, anmelden. Nach der Anmeldung verweilen die Teilnehmer so lange auf dieser Seite, bis der Teacher die Umfrage startet. In diesem Fall findet eine Weiterleitung auf die Seite `quiz.xhtml` statt, die ihrerseits wiederum `quizDialogue.xhtml` einbindet. Nach Abschluss oder Abbruch der Umfrage werden alle Teilnehmer automatisch auf die Seite `takeQuiz.xhtml` zurückgeführt.

Resultate

Alle durchgeführte Umfragen die auf einem QTI Content Package basieren, können auch nach der Beendigung über die Seite `reportsOverview.xhtml` noch aufgerufen werden. Dafür muss zuerst der Teilnehmer und anschliessend das Datum der Ausführung ausgewählt werden.

Studenten

Die sehr einfach gehaltene Seite `studentOverview.xhtml` dient alleine dem Zweck, alle Teilnehmer, die im Augenblick am System angemeldet sind, auszugeben.

Benutzer

Diese letzte Kategorie dient dem User Management. Die Seite `userOverview.xhtml` listet alle in der Datenbank vorhandenen Teilnehmer auf, erlaubt das Zuweisen an eine Gruppe oder das endgültige Löschen (damit verbundene Antworten bleiben jedoch bestehen). Für das Anlegen neuer Benutzer oder Gruppen wird die Seite `createUser.xhtml` verwendet.

Komponenten

Um die Webseiten so zu gestalten, wie sie letztendlich aussehen, bedarf es der Integration einiger Komponenten. Neben Tomahawk, der Bibliothek die in MyFaces mitgeliefert wird, ist vor allem RichFaces die Funktionalität zu verdanken. Während einige Komponenten nur zur stilistischen Verschönerung dienen, wurden andere wegen ihrer AJAX-Fähigkeit integriert. In der folgenden Beschreibung werden alle wichtigen RichFaces Komponenten vorgestellt.

- **a4j:poll** dient dem Aktualisieren von Zielelementen. Diese Komponente wird an zahlreichen Stellen in der Anwendung eingebunden, sei es dass ein Student auf eine

Umfrage wartet, oder sei es dass der Teacher überprüft, wieviele Studenten momentan am System angemeldet sind. In beiden Fällen wird im Hintergrund stets ein Poll durchgeführt, d.h. die Komponente erneuert in einem definierten Intervall die angegebenen Elemente. Wird diese Methode zum Beispiel auf eine Datenliste angewandt, dann befragt diese Liste bei jeder Erneuerung die Geschäftslogik nach dem aktuellen Zustand. Der Benutzer bekommt von diesen regelmässigen Zugriffen nichts mit, da alle Zugriffe asynchron ablaufen und deshalb kein Neuladen der Seite notwendig ist. Wie aus der Erklärung bereits hervorgeht, sind vorallem zwei Attribute essentiell für die Anwendung, das Intervall der Aktualisierung in Millisekunden (*interval*) sowie das zu aktualisierende Ziel (*reRender*). Ein Beispiel dafür könnte folgendermaßen aussehen: `<ajax:poll id="poll" intervall="1000" reRender="poll,data" />`. Das Attribut *id* muss ebenfalls gesetzt sein, da als Ziel neben dem gewünschten Element auch die poll-Komponente selbst neu geladen werden muss, ansonsten wird nur einmal ein poll durchgeführt.

- **rich:progressbar** stellt, wie der Name bereits verrät, einen Fortschrittsbalken dar (Abbildung 2.8). Diese Komponente wird auf der Seite des Teachers während einer Umfrage angezeigt, um den Fortschritt der einzelnen Studenten zu verfolgen. Die Besonderheit der Komponente liegt in der Fähigkeit, sich selbst regelmässig zu aktualisieren. Ähnlich wie bei *ajax:poll* muss dafür ein Intervall in Millisekunden angegeben werden. Das Zielelement jedoch fällt weg, da die Aktualisierung nur auf den Balken selbst angewandt wird. `<rich:progressBar value="#{quiz.value}" intervall="2000" mode="ajax" minValue="0" maxValue="100"/>` zeigt wie ein minimales Beispiel dafür aussehen könnte. Das bereits bekannte Intervall (*interval*) gibt den zeitlichen Abstand der Aktualisierung an und *mode="ajax"* lässt die Komponente asynchron laufen. Schliesslich werden noch drei Werte benötigt, die obere bzw. untere Grenze und der derzeitige Wert. Letzterer wird regelmässig aus der Geschäftslogik ausgelesen (über `#{quiz.value}` wird auf das Feld *value* im Backing-Bean *quiz* zugegriffen).

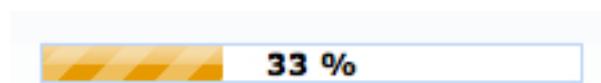


Abbildung 2.8: RichFaces Progressbar

- **rich:fileUpload** erlaubt das Hochladen einer Datei vom Client auf den Server (Abbildung 2.9). Innerhalb der Anwendung wird diese Komponente zweimal verwendet, einerseits beim Erstellen einer QTI Umfrage, andererseits beim Auswäh-

len eines QTI Content Package damit der Teacher die Umfrage im Vorfeld betrachten kann. Wie der Quellcode auszusehen hat, zeigt das folgende Beispiel: `<rich:fileUpload fileUploadListener="#{qtiPlayer.uploadListener}" maxFilesQuantity="1" id="upload" immediateUpload="false" acceptedTypes="*" />`. Das Attribut `fileUploadListener` definiert die Methode in der Geschäftslogik, die den Upload ausführen soll. Sobald der Upload Button getätigt wird, wird diese Methode aufgerufen und beginnt mit dem Datentransfer. Die restlichen Attribute in dem Beispiel dienen der Feinjustierung. Das Attribute `maxFilesQuantity` legt fest, wieviele Dateien für den Upload-Prozess angegeben werden dürfen, in diesem Fall ist die Anzahl auf eine einzige Datei begrenzt. Mit `immediateUpload` kann man festlegen, ob die ausgewählte Datei direkt nach dem Selektieren hochgeladen werden soll, d.h. es wird kein Knopfdruck benötigt. Schliesslich darf durch `acceptedTypes` noch die Endung der hochzuladenden Datei eingeschränkt werden (z.B. jpg, gif etc.).

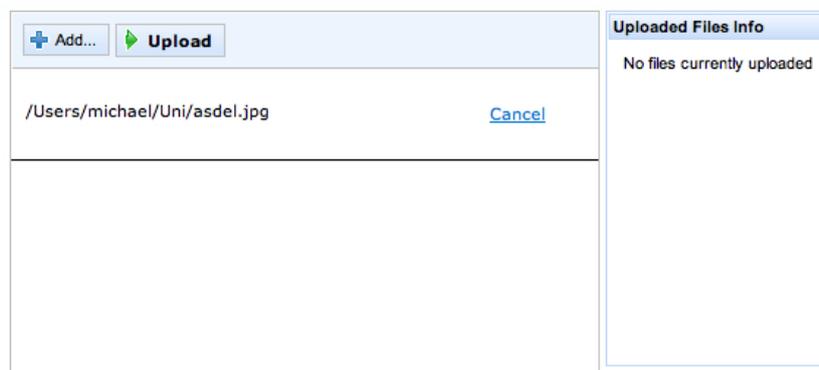


Abbildung 2.9: RichFaces Upload

- **a4j:commandButton** ähnelt sehr der Variante von MyFaces, erweitert diese jedoch um das AJAX-Attribut `reRender`. Während die MyFaces Variante nach dem Betätigen des Knopfes die Seite neu lädt (entspricht dem typischen JSF Lifecycle), spricht die RichFaces Alternative die Geschäftslogik über einen asynchronen Aufruf an und aktualisiert anschliessend das Zielelement. Das folgende Beispiel soll die Verwendungsweise verdeutlichen: `<a4j:commandButton value="Next" actionListener="#{quizControl.nextQuestion}" reRender="quizForm" />`. Der `value` legt die Beschriftung des Knopfes fest, der `actionListener` definiert die aufzurufende Methode im dahinterstehenden Bean und das `reRender` gibt die zu aktualisierende Komponente an. Im Rahmen dieser Anwendung wird der `a4j:commandButton` für die Visualisierung der manuellen Tests auf Seite der Studenten benutzt. Sobald ein Student seine individuelle Umfrage startet, erscheint ein Popup-Fenster

mit der ersten Frage. Nach gegebener Antwort kann man über den Next-Button die nächste Frage laden. Und genau dieser Next-Button wird durch den RichFaces CommandButton realisiert. Würde man an dieser Stelle die MyFaces Version wählen, dann würde die Funktionsweise durch das erneute Laden der Seite gestört werden (siehe Abschnitt 2.1.3).

Alle anderen in der Anwendung integrierten RichFaces Komponenten, die sich vor allem durch den für RichFaces bekannten Style auszeichnen, werden in Tabelle 2.3 kurz beschrieben.

Komponente	Funktion
rich:dataTable	Tabelle
rich:modalPanel	Popup
rich:panel	Fenster mit Überschrift
rich:panelMenu	Faltbares Panel
rich:spacer	Trennzeichen
rich:tabPanel	Panel mit Reitern
rich:inplaceInput	Eingabefeld wenn ausgewählt, ansonsten Textdarstellung

Tabelle 2.3: RichFaces Komponenten

2.4.4 Geschäftslogik

Während im vorherigen Kapitel alle Webseiten vorgestellt wurden, widmet sich dieser Teil der Geschäftslogik, die hinter den einzelnen Seiten steckt und die eigentliche Funktionalität zur Verfügung stellt. Ähnlich wie bei den Webseiten sind die Java Klassen in diverse Kategorien unterteilt:

Package	Java Klassen
Nachrichten	AnnouncementOverview.java
Aufgaben	AssignmentOverview.java, CreateAssignment.java
Beans	AnnouncementBean.java, AnswerBean.java, AssignmentBean.java, BaseBean.java, QtiBean.java, QuestionBean.java, QuizBean.java, ReportBean.java, UserBean.java, UsergroupBean.java

Filter	HibernateFilter.java, LoginCheckFilter.java, SessionCleanupFilter.java
Anmeldung	Login.java
QTI	QTIPlayer.java
Umfragen	QuizControl.java, QuizOverview.java
Resultate	ReportOverview.java
Services	DAOService.java, HibernateService.java, UserService.java
Studenten	StudentOverview.java
Benutzer	UserOverview.java
Diverses	Constants.java

Tabelle 2.4: Packages und Java Klassen

Alle in Tabelle 2.4 aufgelisteten Java Klassen werden in den folgenden Abschnitten im Detail erläutert.

Nachrichten

Die Klasse *announcementOverview.java* ermöglicht dem Teacher das Verfassen, Löschen und Anpassen von Nachrichten, die allen Studenten beim nächsten Login am System angezeigt werden.

Aufgaben

Die Klasse *AssignmentOverview.java* stellt Methoden für die Bearbeitung von Aufgaben zur Verfügung. Es können einem Assignment Gruppen zugewiesen und diverse Stati gesetzt werden. Neue Assignments können mit Hilfe der Klasse *CreateAssignment.java* erstellt werden.

Beans

Die Klassen in diesem Package beinhalten keine Geschäftslogik und dienen nur zur Speicherung von Daten.

Filter

Das Package *filter* besteht aus den drei Klassen *HibernateFilter*, *LoginCheckFilter* und *SessionCleanupFilter*. Der Aufbau eines Filters wird im Detail in Abschnitt 2.4.8 beschrieben.

Anmeldung

In der Klasse *Login.java* wird der An- und Abmeldevorgang am System abgewickelt.

QTI

QTIPlayer.java dient der Darstellung von Fragen im QTI Format. Die dazu benötigte Kommunikation mit dem *player* wird in dieser Klasse abgehandelt.

Umfragen

In der Klasse *QuizControl.java* liegt die komplette Logik, die für die Durchführung einer Umfrage benötigt wird. Über die Klasse *QuizOverview.java* können sowohl bestehende Umfragen betrachtet als auch neue erstellt werden.

Resultate

Die Ergebnisse von Umfragen sollen auch zu einem späteren Zeitpunkt noch zur Verfügung stehen. Deshalb bietet die Klasse *ReportOverview.java* sämtliche Methoden zum

Laden der Resultate von QTI und manuellen Umfragen.

Services

Die Klasse *UserService.java* stellt dem Login einige Methoden zur Verfügung und speichert alle angemeldeten Benutzer. Die anderen beiden Service Klassen, *DAOService.java* und *HibernateService.java*, werden im folgenden Kapitel 2.4.5 im Detail erklärt.

Studenten

Über die Klasse *StudentOverview.java* kann die Liste aller am System angemeldeten Studenten abgefragt werden.

Benutzer

Sämtliche Operationen, die den Benutzer betreffen, werden in der Klasse *UserOverview.java* abgehandelt.

Diverses

Die Klasse *Constants.java* enthält nur statische Variablen, die in den anderen Klassen verwendet werden. Auf diese Weise können Namensänderungen, zum Beispiel für Pfade, zentral durchgeführt werden.

2.4.5 Datenbank in Verbindung mit Hibernate

In Abschnitt 2.2.6 ist bereits die Konfiguration von Hibernate besprochen worden, an dieser Stelle soll jetzt der implementierungstechnische Teil nachfolgen. Die beiden Klassen *HibernateService* und *DAOService* übernehmen das Transaktionsmanagement, wobei die Klasse *HibernateService* das Session- und Transactionhandling übernimmt während

die Klasse *DAOService* die Datenbank Schnittstelle für alle anderen Klassen darstellt. In Listing 2.6 findet man alle relevanten Felder und Methoden der Klasse *HibernateService* wieder.

```
1 public class HibernateService {
2
3     private static final SessionFactory sessionFactory;
4     private static Configuration configuration;
5     private static final ThreadLocal threadSession = new ThreadLocal();
6     private static final ThreadLocal threadTransaction = new ThreadLocal();
7
8     static {
9         try {
10            configuration = new Configuration();
11            sessionFactory = configuration.configure().buildSessionFactory();
12        } catch (Throwable ex) {
13            throw new ExceptionInInitializerError(ex);
14        }
15    }
16
17    public static Session getSession();
18    public static void closeSession();
19    public static void beginTransaction();
20    public static void commitTransaction();
21    public static void rollBackTransaction();
22
23 }
```

Listing 2.6: Auszug aus HibernateService.java

Die Klasse *DAOService* besteht aus einer Vielzahl von Methoden, die sich alle der Klasse *HibernateService* bedienen und den Backing Beans Zugriff auf die Datenbank verschaffen. Die elementaren Methoden werden anhand von Listing 2.7 veranschaulicht.

```
1 public class DAOService {
2
3     public DAOService() {
4         HibernateService.beginTransaction();
5     }
6
7     public void saveOrUpdate(Object o) {
8         HibernateService.getSession().saveOrUpdate(o);
9     }
}
```

```

10
11 public void save(Object o);
12 public void merge(Object o);
13 public void delete(Object o);
14 public void initialize(Object o);
15 public void close();
16 public void flush();
17
18 ...
19
20 public UserBean getUser(String id) {
21     return (UserBean)HibernateService.getSession().load(UserBean.class, id);
22 }
23
24 public List<QuizBean> getQuizzes() {
25     return HibernateService.getSession().createQuery("from QuizBean").list();
26 }
27
28 }

```

Listing 2.7: Auszug aus DAOService.java

Die meisten angeführten Methoden sind bereits aus dem Hibernate Lifecycle aus Abschnitt 2.2.6 bekannt und bilden die grundlegenden Operationen, die innerhalb einer Session ausgeführt werden können. Hibernate erlaubt aber auch eigene Abfragen, die in der Hibernate Query Language (HQL) verfasst werden, zu gestalten. In der letzten Methode *getQuizzes()* in Listing 2.7 wird mittels dem Aufruf *createQuery("from QuizBean")* eine Hibernate Query erzeugt, wobei das Statement *from QuizBean* der HQL Syntax entsprechen muss.

2.4.6 Rollen und Rechte

Wenn ein neuer Benutzer angelegt wird, muss ihm unter anderem auch eine Rolle zugewiesen werden. Das System sieht vier verschiedene Rollen vor, jede mit eigenen Rechten behaftet. Der *ADMINISTRATOR* hat volle Zugriffsrechte, er darf alle Seiten betreten und kann neue Benutzer anlegen und bestehende verwalten. Weiters kann er Umfragen erstellen und durchführen, Ergebnisse auswerten und Meldungen verfassen. Der *TEACHER* hat beinahe dieselben Rechte wie der *ADMINISTRATOR*, mit dem kleinen

Unterschied, dass er keine Benutzer mit der Rolle *TEACHER* oder *ADMINISTRATOR* anlegen darf, das ist dem *ADMINISTRATOR* vorbehalten. Ansonsten ist der *TEACHER* in keiner Hinsicht eingeschränkt, er kann alle Seiten betreten und die volle Funktionalität geniessen. Die dritte verfügbare Rolle, der *STUDENT*, hat nur mehr eine begrenzte Auswahl an Interaktionsmöglichkeiten. Er kann einerseits Nachrichten auf der Startseite lesen und andererseits an Umfragen teilnehmen. Die vierte und letzte Rolle trägt den Namen *VISITOR*. Diese Rolle kann, im Gegensatz zu den anderen, keinem Benutzer zugewiesen werden, sondern ist für anonyme Umfragen gedacht. Dazu öffnet ein *TEACHER* eine Umfrage und weist dieser die Gruppe *VISITOR* zu. Anschliessend können Benutzer über einen bestimmten Link die Plattform anonym betreten und an dieser Umfrage teilnehmen. Jeder Benutzer, der über den *VISITOR* Link die Seite betritt, bekommt den Namen user gefolgt von einer Zahl, die bei 0 beginnend für jeden neuen Benutzer um eins erhöht wird.

2.4.7 Logging

Die fehlerfreie Funktionsweise der Anwendung soll nicht erst nach der Testphase gegeben sein, sondern bereits während der Implementierung erreicht werden. Mit Hilfe des Loggings kommt man dem Ziel schon einen Schritt näher. Es können im Quellcode Befehle gesetzt werden, die eine Ausgabe in eine definierte Logdatei oder auf der Konsole erzwingen. Mit dieser Methode lassen sich etwa Resultate von Operationen oder Parameter auslesen, das gibt dem Programmierer die nötige Transparenz, so dass er auf Grund der Ergebnisse Fehler bereits frühzeitig eingrenzen kann. Wie bereits in Abschnitt 2.2.11 angemerkt wurde, handelt es sich bei der verwendeten Logging Implementierung um log4j von Apache. In allen Klassen wird unter anderem ein log4j Logger initialisiert, der alle Log-Einträge für die aktuelle Klasse vornimmt. Listing 2.8 zeigt, wie so ein Eintrag aussehen kann.

```
1 private static final Logger logger = Logger.getLogger(Login.class);
```

Listing 2.8: Definieren eines log4j Loggers (Auszug aus der Klasse Login.java)

Nachdem der logger definiert wurde, kann er an jeder Stelle innerhalb der Klasse verwendet werden. Listing 2.9 zeigt, wie ein typischen Beispiel aussehen könnte.

```

1 if (logger.isDebugEnabled()) {
2     logger.debug("User has successfully logged in");
3 }

```

Listing 2.9: Logging Beispiel

Das Ergebniss kann anschliessend in der Logdatei betrachtet werden, für das obige Beispiel würde die Ausgabe folgendermaßen aussehen:

```

2008-10-21 09:20:37,703 [http-8080-4] DEBUG login.Login User has successfully logged
in (Login.java:87)

```

Dieser kurze Eintrag kann bei der Fehlersuche sehr viel Zeit ersparen. Ein Blick in die Logdatei verrät sofort, ob der User eingeloggt ist. Ist das nicht der Fall, muss der Fehler vor dem Statement aufgetreten sein und kann so eingegrenzt werden.

Eine weitere nennenswerte Eigenschaft von Log4j ist die Möglichkeit, mit Hilfe der Abfrage *logger.isDebugEnabled()* festzustellen, ob man sich im Debug Modus befindet oder nicht. Trifft letzterer Fall zu, werden alle Parameter im Log-Aufruf ignoriert um Laufzeit zu sparen.

2.4.8 Filter

Java Servlet Filter werden in JSF benutzt, um zwischen zwei Lifecycle Zyklen gewünschte Methoden auszuführen, d.h. bevor eine Seite geladen wird, werden alle Filter aufgerufen. Wenn etwa ein Benutzer eine Seite betreten will, überprüft der LoginCheckFilter vorab, ob der Benutzer am System angemeldet ist und leitet ihn gegebenenfalls auf die Login Seite weiter.

Java stellt für Filter drei verschiedene Interfaces zur Verfügung: *java.servlet.FilterConfig*, *java.servlet.FilterChain* und *java.servlet.Filter*. Jeder selbst definierte Filter muss das Interface *java.servlet.Filter* und dessen Methoden (*init*, *doFilter*, *destroy*) implementieren. Die Methode *init* dient der Initialisierung der benötigten Parameter und benutzt dafür das Interfaces *java.servlet.FilterConfig*. Alle Parameter, die der Filter auslesen will, müssen zuvor im Deployment Descriptor (web.xml) definiert werden.

Die Methode *doFilter* beinhaltet die eigentliche Logik. Das Interface *java.servlet.FilterChain* erlaubt das Hintereinanderschalten mehrerer Filter, wobei die genaue Reihenfolge im De-

ployment Descriptor festgelegt wird.

Abschliessend wird die Methode *destroy* aufgerufen, in der, falls nötig, reservierte Ressourcen wieder freigegeben werden, im Normalfall bleibt diese Methode aber leer.

Im Rahmen dieser Anwendung kommen die folgenden drei Filter zum Einsatz:

- Der sehr einfach gehaltene **HibernateFilter** hat nur die Aufgabe, nach jedem Zyklus ein Commit Richtung Datenbank durchzuführen und schliesslich die Datenbankverbindung zu schliessen, damit keine Verbindungen unnötig offen bleiben.
- Der **LoginCheckFilter** soll verhindern, dass fremde bzw. nicht eingeloggte Benutzer eine Seite betreten können. Ruft jemand eine andere Seite als die Login-Seite auf, dann überprüft der Filter, ob sich derjenige schon am System angemeldet hat. Ist das der Fall, darf er die Seite betreten, ansonsten wird er auf die Login-Seite weitergeleitet.
- Der komplexeste der drei Filter ist der **SessionCleanupFilter**. Er hat die Aufgabe, die Geschäftslogik einer Webseite (i.d.R. das entsprechende Backing-Bean) aus der Session zu nehmen, nachdem die Seite verlassen wurde. Leider gibt es einige Ausnahmen zu beachten, da einige Backing-Beans von mehreren Benutzern angesprochen werden und deshalb nicht so ohne Weiteres entfernt werden können (siehe Abschnitt 2.6.4)

2.4.9 Einstellungen durch Properties

Die Anwendung sieht drei Konfigurationsdateien vor, mit deren Hilfe gewisse Einstellungen vorgenommen werden können:

- Die Datei *hibernate.cfg.xml* ist zwar der Endung nach keine .properties Datei, dennoch können über die darin enthaltenen Einträge zahlreiche Hibernate Konfigurationen vorgenommen werden. So können zum Beispiel der Datenbanktyp (HSQLDB, MySQL etc.), die Datenbank URL, Username, Passwort entsprechend eingestellt werden (siehe Abschnitt 2.2.6).
- Alle Einstellungen, die das Logging mit log4j betreffen, können über die Datei *log4j.properties* vorgenommen werden. Dazu zählen Einträge wie zum Beispiel der Pfad der Logging Datei, die Sicherheitsstufe bei der gelogged werden soll oder das Aussehen der Log Einträge (siehe Abschnitt 2.2.11).

- In der Datei `configuration.properties` lassen sich Pfade festlegen für
 1. die gesamte Anwendung (*applicationPath*)
 2. den Upload Ordner (*uploadPath*)
 3. den Ordner, der die XSL Templates beinhaltet (*xslPath*)
 4. den Ordner für XML Reports (*reportPathXml*)
 5. den Ordner für HTML Reports (*reportPathHtml*)
 6. den playr (*playrUrl*)

2.4.10 Stylesheets mit CSS

Bei modernen Webanwendungen ist es üblich, Formatierungen und Style-Elemente aus den HTML Seiten zu entfernen und in eine eigene Datei auszulagern. Alle Einträge in dieser Datei mit der Endung `.css` werden über die deklarative Stylesheet-Sprache Cascading Style Sheet (CSS) verfasst. Der Vorteil dieser Methode liegt in der strikten Trennung zwischen der optischen Gestaltung und dem eigentlichen Inhalt. Auf diese Weise ist es möglich, das Aussehen für verschiedene Ausgabemedien anzupassen, indem man mehrere Stylesheets erstellt und für jeden Zweck das gewünschte verwendet.

Ein weiterer Vorteil von CSS liegt in der Code Einsparung. Wurden vorher alle Komponenten einzeln mit einem Style versehen, können mit CSS Styles, die auf mehreren Seiten verwendet werden, zu Gruppen zusammengefasst und an den entsprechenden Orten per Referenz eingebunden werden.

2.5 Die Datenbank

Während der Entwicklungsphase hat sich das Datenbankschema mehrmals verändert, weshalb anfangs HSQLDB als Datenbank gewählt wurde (siehe Abschnitt 2.2.7). HSQLDB erlaubt sowohl die Generierung der Tabellen im Hauptspeicher, d.h. es werden keine Daten auf der Festplatte angelegt und nach dem Beenden der Anwendung werden die Daten wieder entfernt, als auch die Speicherung der Tabellen in einer Datei. Nachdem das Datenbankschema fixiert wurde, erfolgte der Datenbank Umstieg auf MySQL. In der Tabelle 2.5 werden alle verwendeten Tabellen mit einer kurzen Erklärung aufgelistet.

Tabelle	Inhalt
announcementbean	Announcements
assignments	Assignments
answers	Antwort-Labels, die in einer Frage angegeben werden
choices	Auswahlmöglichkeiten von Dropdown Listen
group_assignment	Einem Assignment wird eine Gruppe zugeteilt
group_quiz	Einer Umfrage wird eine Gruppe zugeteilt
quizzes	Umfragen
questions	Fragen, die in Umfragen verwendet werden
quiz_questions	Einer Umfrage werden Fragen zugewiesen
question_answer	Einer Frage werden Antwortmöglichkeiten zugewiesen
reports	Umfragen-Ergebnisse
user_group	Einer Usergroup werden User zugeteilt
userbean	Alle registrierten Benutzer
usergroups	Usergruppen

Tabelle 2.5: Datenbanktabellen

2.6 Testen der Anwendung

Der Entwicklungsprozess wurde von kontinuierlichen Testläufen begleitet, welche die Funktionalität und Fehlerfreiheit der Applikation garantieren sollen. Aber auch nach der abgeschlossenen Entwicklung gilt es weitere Tests durchzuführen, um versteckte Fehler ausfindig zu machen. Einfach darauf loszutesten führt dabei meist nicht zu dem gewünschten Ergebnis, das Erstellen eines Testplans hingegen schon. In diesem Testplan versucht man alle Interaktionsmöglichkeiten abzudecken, d.h. alle Möglichkeiten die dem Benutzer innerhalb der Anwendung angeboten werden müssen berücksichtigt werden, auch wenn die Abfolge nicht immer logisch erscheint. Wird während dem Abarbeiten des Plans ein Fehler gefunden, dann wird der Testprozess abgebrochen und der Fehler beseitigt. Anschliessend beginnt das Prozedere von vorne, dabei müssen auch bereits abgeschlossene Testfälle erneut durchgeführt werden, um Regressionen auszuschliessen. Es könnte ja durch das Beheben des Fehlers ein neuer entstanden oder ein alter wiederaufgelebt sein. Obwohl es möglich wäre die Anwendung mit einem Mehraufwand durch automatisierte Verfahren testen zu lassen, war es in diesem Fall ausreichend, alle Tests manuell durchzuführen. Dazu wurden alle Webseiten, (AJAX-) Komponenten, etc. einzeln und per Hand auf ihre Funktionalität überprüft.

In den folgenden Abschnitten werden die einzelnen Aspekte der Anwendung, die getestet wurden, beschrieben.

2.6.1 Kompatibilitätstests

Die Anwendung wurde während der Entwicklung mit dem Browser Mozilla Firefox getestet, sie sollte aber in der finalen Version in den gängigsten Browsern betrieben werden können. Aus diesem Grund mussten alle Webseiten in den folgenden Browsern getestet bzw. betrachtet werden:

- Mozilla Firefox 3.0.5
- Microsoft Internet Explorer 7.0.5730.11
- Google Chrome 1.0.154.43
- Apple Safari 3.2.1

Insbesondere die Funktionalität aller RichFaces Komponenten war ein wichtiges Testkriterium, da sie u.a. den Ablauf einer Umfrage realisieren.

2.6.2 Kommunikation

Der grösste und wichtigste Teil der Anwendung, die Kommunikation zwischen Teacher und Student, ist zugleich auch der fehleranfälligste. Die Hauptfehlerquelle liegt in den definierten Variablen und deren Abfragen. Für jeden Zustand, in dem sich ein Student oder Teacher befinden kann, gibt es eine begrenzte Anzahl an Interaktionsmöglichkeiten. Um festzustellen, welche Aktion durchgeführt werden darf, müssen die eben erwähnten Abfragen durchgeführt werden. Ist eine der Variablen falsch belegt oder liefert eine Abfrage einen falschen Wert, dann werden dem Benutzer inkorrekte Informationen angezeigt und die Funktionalität der Anwendung ist nicht mehr gegeben.

2.6.3 AJAX-Komponenten

Im Rahmen der Kommunikations-Tests werden auch die AJAX Komponenten von RichFaces auf die korrekte Ausführung und Browser-Kompatibilität überprüft. Besonderes Augenmerk soll in diesem Zusammenhang auf folgende Komponenten gerichtet werden:

Die **Progressbar** muss den aktuellen Fortschritt kontinuierlich ermitteln und entsprechend anzeigen. Dabei ist es wichtig, dass sowohl der Start- als auch der Endwert richtig gesetzt sind, da ansonsten das Resultat verzerrt wird. Des Weiteren gilt es den aktuellen Wert zu überprüfen, der von der entsprechenden Geschäftslogik ermittelt und auf der Seite als aktueller Fortschritt angezeigt wird. Abschliessend muss noch die Grundfunktionalität der Komponente, das automatische Aktualisieren, verifiziert werden.

Die **Polling**-Komponente realisiert einige der wichtigsten Funktionen in der Anwendung und erlaubt die Aktualisierung von Elementen ohne die komplette Seite neu laden zu müssen. Aus diesem Grund ist es sehr wichtig, diese Komponente auf allen sie verwendenden Seiten zu testen. Dabei gilt es zwei Punkte zu überprüfen, die Aktualisierung in einem bestimmten Intervall und das erneute Rendern des Zielelements. Gerade beim letzteren Punkte ist Vorsicht geboten, da sich die Aktualisierung nicht auf alle beliebigen Elemente anwenden lässt, sondern auf einige Komponenten begrenzt ist.

Commandbuttons werden nicht nur von MyFaces mitgeliefert, sondern sind auch in der RichFaces Bibliothek verfügbar, weshalb die AJAX Funktionalität gesondert getestet werden muss. Der Vorteil dieser Buttons liegt darin, dass damit Funktionen der Geschäftslogik aufgerufen werden können ohne einen Page Reload zu verursachen. Wie auch bei der Polling-Komponente wird nach dem Aufruf ein Zielelement neu gerendert, dabei müssen ebenfalls dieselben Regeln beachtet werden.

Der **Upload-Dialog** spielt eine elementare Rolle in der Anwendung, da er das asynchrone Hochladen von QTI Content Packages ermöglicht. Diese Komponenten hat mehrmals Schwierigkeiten bereitet (siehe Kapitel 2.7), weshalb genaueres Testen der folgenden Punkte unabdingbar war:

- Korrekte Darstellung des Upload-Dialogs
- Betätigen des Add-Buttons öffnet den Dialogue
- Ausgewähltes QTI Content Package befindet sich nach dem Hochladen im Ordner Upload

2.6.4 Filter

Die beiden Filter **SessionCleanupFilter** und **LoginCheckFilter** realisieren essentielle bzw. sicherheitskritische Funktionen in der Anwendung und müssen deshalb entsprechend genau getestet werden.

Für den **LoginCheckFilter**-Test ist es ausreichend, einen Quereinstieg in die Anwendung durchzuführen, d.h. es wird nicht die Login-Seite aufgerufen sondern eine beliebig andere. Ist der Benutzer noch nicht eingeloggt, müsste er sofort automatisch auf die Login-Seite weitergeleitet werden. Falls er jedoch schon am System angemeldet ist, darf er die Seite betreten, d.h. es findet keine Umleitung statt.

Der **SessionCleanupFilter**-Test ist etwas anspruchsvoller, da es einige Ausnahmen zu beachten gilt. Am besten lässt sich die korrekte Funktionalität des Filters mit Log-Einträgen nachvollziehen. Sobald eine Seite verlassen wird, soll die dahinterstehende Geschäftslogik in Form eines Backing-Beans aus der Session genommen werden. Handelt es sich bei dem Backing-Bean aber um die Klasse `quizControl.java`, müssen spezielle Regeln beachtet werden. Während einer Umfrage kann das vorzeitige Entfernen des Backing-Beans aus der Session Konsequenzen nach sich ziehen, in den meisten Fällen das fehlerhafte Abbrechen der Umfrage mit einer Exception. Um dieses Szenario zu vermeiden, überprüft der Filter zuerst, ob eine Umfrage offen ist und nimmt das Backing-Bean erst nach deren Abschluss aus der Session.

2.6.5 QTI Player

Der QTI Player stellt neben der in Abschnitt 2.6.2 beschriebenen Kommunikation die zweite grosse Fehlerquelle dar und bedarf folgender dreier Testphasen:

- Testphase 1: Kommunikation zwischen Anwendung und player
- Testphase 2: Korrekte Darstellung aller 16 Fragetypen
- Testphase 3: Überprüfen der JavaScript Funktionen

Ad Testphase 1: Die Kommunikation zwischen der Anwendung und dem player läuft in mehreren Schritten ab, weshalb alle Zwischenergebnisse kontrolliert werden müssen. Die einzelnen Schritte mit den erwarteten Resultaten werden in Tabelle 2.6 beschrieben.

	Methode	Resultat
1.	loadPaths()	Alle Pfade werden gesetzt
2.	getNewPlayrId()	Eine neue Session Id wird angefordert
3.	getFirstPage()	Content Package wird übermittelt/validiert und erste Seite zurückgeliefert
4.	checkForEnd()	Überprüft ob das Ende der Umfrage erreicht wurde
5.	replaceConstants()	Überarbeitet den vom playr gelieferten xhtml Code
6.	play()	Alle Parameter müssen ausgelesen und an den playr gesendet werden

Tabelle 2.6: Methoden und Resultate des QTIPlayer

Ad Testphase 2: Die aktuelle IMS QTI Spezifikation sieht 16 Fragetypen vor, die alle vom playr unterstützt werden, und daher auch getestet werden müssen. Manche Fragetypen bauen zum Beispiel auf Applets auf. Wird das Applet nicht korrekt gestartet oder ist ein Pfad fehlerhaft, dann bleibt die Seite leer und die Umfrage kann nicht mehr fortgesetzt werden. In vielen Fällen werden auch Bilder eingebunden um die Frage zu verdeutlichen. Befindet sich das Bild nicht im richtigen Ordner oder der Pfad ist falsch gesetzt, dann wird nur die Frage in schriftlicher Form dargestellt und das könnte das Verständnis beeinflussen.

Ad Testphase 3: Der playr bedient sich bei der Anzeige einer Frage diverser JavaScript Funktionen, die die korrekte Handhabung garantieren sollen. Wenn etwa eine Frage drei Checkboxen anbietet, wobei zwei davon ausgewählt werden sollen, dann kann mittels entsprechender JavaScript Funktion eine Fehlermeldung angezeigt werden, sobald der Benutzer versucht alle drei Boxen zu markieren. Wie das Beispiel zeigt, ist es daher wichtig, alle JavaScript Funktionen, die zur Verfügung stehen, zu testen.

2.6.6 Datenbankzugriffe

Ein weiterer Punkt auf der Testliste ist der Datenbankzugriff über Hibernate. Wie bereits in 2.2.6 angemerkt wurde, übernimmt Hibernate die Aufgabe der Persistierung. Obwohl einem diese Technologie das Verfassen von Statements (SQL Befehlen) abnimmt und somit viel Arbeit und Zeit erspart, bedarf es einiger Einarbeitungszeit, bis Hibernate korrekt läuft. Zahlreiche Konfigurationsmöglichkeiten erschweren dem Benutzer die Handhabung, weshalb gerade anfangs oftmals nicht das gewünschte Resultat erzielt wird,

sei es, dass ein Datensatz nicht in der Datenbank gespeichert wird, oder sei es, dass benötigte Daten nicht sofort geladen werden. Aus diesem Grund ist es unumgänglich, auch die Datenbank und alle Zugriffe darauf in die Tests miteinzubeziehen. Für MySQL Datenbankabfragen hat sich der MySQL Query Browser bewährt. Damit lässt sich nach jeder Transaktion sofort überprüfen, ob die Daten wie gewünscht in der Datenbank gespeichert oder gelöscht wurden.

2.6.7 Rollen und Rechte

Abschliessend müssen noch die einzelnen Rollen mit den jeweiligen Rechten getestet werden. Wie schon angemerkt, sieht das System vier verschiedene Benutzertypen vor: Administrator (ADMIN), Lehrer (TEACHER), Student (STUDENT) und Besucher (VISITOR). Auf jeder besuchten Webseite gibt es im Quellcode Abfragen, welchem der genannten Typen der angemeldete Benutzer entspricht. Je nachdem welche Rolle dem Benutzer zugesprochen wurde, kann er auf der Seite mehr oder weniger betrachten. Ein Administrator kann beispielsweise neue Benutzer anlegen, dem Student bleibt diese Option aber verwehrt. Deshalb müssen alle Webseiten aus der Sicht aller Benutzertypen betrachtet werden, um zu verifizieren, dass die Rechte auf der jeweiligen Seite korrekt gesetzt wurden.

2.7 Probleme während der Entwicklung und deren Lösungen

Wie bereits aus Kapitel 2.6 hervorgeht, sind während der Implementierung zahlreiche Probleme aufgetreten, die in drei Kategorien unterteilt werden können: technologische, implementierungstechnische und QTI-bezogene Probleme. Im Rahmen dieses Kapitels werden diese drei Kategorien im Detail behandelt und Lösungswege beschrieben.

2.7.1 Technologische Probleme

In diese Kategorie fallen alle Probleme und Schwierigkeiten, die während der Implementierung auf Grund von Technologien aufgetreten sind, sei es durch Konflikte zwischen Komponentenbibliotheken, oder sei es durch Versionsabhängigkeiten zwischen verschiedenen Technologien.

Zu Beginn der Entwicklung wurde neben der Bibliothek Tomahawk vor allem Trinidad verwendet, da diese AJAX-Unterstützung anbot und sich in die Apache Reihe nahtlos einfügte. Zu einem späteren Zeitpunkt wurde RichFaces zu den Komponentenbibliotheken hinzugefügt, da es weitere Funktionalitäten anbot, ein ansprechendes Design mitbrachte und vor allem sehr einfach zu bedienen war. Obwohl RichFaces und Trinidad von den verfügbaren Komponenten her sehr ähnlich waren, zeichnete sich Trinidad durch die Progressbar aus, die RichFaces zu diesem Zeitpunkt noch nicht in sein Repertoire aufgenommen hatte. Je mehr Komponenten der beiden Bibliotheken in die Anwendung integriert wurden, desto mehr Konflikte und Komplikationen traten auf. Schliesslich wurde Trinidad aus der Anwendung entfernt, da sich RichFaces als die mächtigere Bibliothek etablierte. Mittlerweile enthielt die aktuelle Version von RichFaces auch eine Progressbar, weshalb Trinidad nun ohnehin obsolet war. Leider brachte die Aktualisierung von RichFaces neue Probleme mit sich, da nach dem Einfügen der neuen Version die Anwendung nicht mehr starten wollte. Grund dafür war die Inkompatibilität zu den veralteten anderen Technologien, denn RichFaces setzte die neueste Version von MyFaces voraus, welche wiederum die nächste Java- und Tomcat-Generation benötigte. Es bedurfte einiger Testläufe bis das System mit allen Konfigurationen wieder korrekt startete.

Ein weiteres Problem, das wohl auch in diese Kategorie fällt, ist die unterschiedliche Darstellung in den verschiedenen Browsern. Die Anwendung wurde für Firefox entwickelt, sollte aber auch in alternativen Browsern betrachtet werden können. Mit Microsoft Internet Explorer und Google Chrome konnten alle Seiten dargestellt werden, wenn auch mit ein paar grafischen Unterschieden, da nicht jeder Browser alle Style-Elemente gleich interpretiert. Es gab allerdings einige Probleme mit RichFaces Komponenten, im Speziellen mit dem Upload Dialog. Mit einer früheren Version der Komponentenbibliothek wollte der Upload Dialog in Chrome nicht erscheinen, mit dem Internet Explorer wurde die gewünschte Datei nicht hochgeladen. In der finalen Version der Anwendung konnten diese Inkompatibilitäten nicht mehr reproduziert werden.

2.7.2 Implementierungstechnische Probleme

Die meisten Probleme, die während der Entwicklung aufgetreten sind, fallen in diese Kategorie. In der folgenden Aufzählung werden die wichtigsten und problematischsten Fälle erläutert.

- Der **Workflow während einer Umfrage** ist eigentlich vorgegeben und sollte auch eingehalten werden. Nichtsdestotrotz müssen auch alle Fälle simuliert werden, die zwar nicht logisch erscheinen und vielleicht auch niemals eintreten, denn ein unerwartetes Ergebnis kann eine komplette Umfrage beeinträchtigen (siehe Abschnitt 2.6). Neben dem Testen der verschiedenen Szenarien ist es vor allem auch wichtig, die Simulation mit mehreren Teilnehmern durchzuführen. Während der Implementierung wurde zu Testzwecken immer ein Teacher und ein Student angenommen, denn das war ausreichend um die Kommunikation zwischen den beiden zu überprüfen. In der abschliessenden Testphase sollte dann das reale Szenario mit einem Teacher und mehreren Studenten nachgebildet werden. Bereits bei zwei angemeldeten Studenten haben sich innerhalb einer Umfrage die ersten Fehler bemerkbar gemacht. Zum Beispiel fand ein Student seine Fragen ausgefüllt vor, da ein anderer Student die Umfrage schneller ausgefüllt und die Geschäftslogik die Ergebnisse automatisch gesetzt hatte. Nachdem diese Problematik beseitigt war, ergab die Analyse der generierten Reports, dass die Umfrage aller Studenten dasselbe Ergebnis lieferte. Dieser Fehler, der ebenfalls in der Geschäftslogik lag, hatte zur Folge, dass jedes Ergebnis überschrieben wurde, sobald ein anderer Student dieselbe Frage beantwortet hatte. Letztendlich konnte auch dieses Problem behoben werden, so dass alle weiteren Umfragen fehlerfrei durchliefen.
- Eine weitere Fehlerquelle bildeten die **Filter**, insbesondere der *SessionCleanupFilter*. Die Aufgabe eines Filters liegt darin, dass er ausgeführt wird, bevor der JSF Lifecycle aktiv wird (siehe Abschnitt 2.4.8). Der *SessionCleanupFilter* soll alle Backing Beans, die nicht mehr benötigt werden, aus der Session nehmen, bevor die nächste Seite angezeigt wird. In den meisten Fällen arbeitet der Filter wie gewünscht, nur das Durchführen von Umfragen bildet eine Ausnahme. Die gesamte Logik, die für die Kommunikation zwischen Teacher und Students verantwortlich ist, befindet sich in der Klasse *quizControl*. Würde man den *SessionCleanupFilter* nicht adaptieren, dann würde er die Klasse sofort aus der Session nehmen, sobald ein Student die Umfrage beendet hat. Die Konsequenzen dieser Aktion reichen von

nicht reagierenden Umfragen (der anderen Teilnehmer) bis hin zum Auftreten von Exceptions. Deshalb wird die Klasse *quizControl* zukünftig vom Filter übergangen, d.h. sie wird in keinem Fall aus der Session genommen.

- Während der Implementierung des Umfrage-Aspekts, haben sich einige **Eigenheiten der RichFaces Komponenten** herauskristallisiert, im Speziellen des *reRender* Attributs. Nicht jedes Zielelement kann für die asynchrone Aktualisierung verwendet werden. In den meisten Fällen bietet sich eine eigene AJAX Umgebung (*ajax:region*) an, in die man das Zielelement einbettet und dessen ID beim *reRender* Attribut angegeben wird. Allerdings muss auch beachtet werden, welche Komponenten sich in der AJAX Umgebung befinden. Während der Implementierung hat sich ein spezielles Szenario ergeben, in dem eine Region, die eine gewissen Anzahl von Progressbars umfasst, definiert und kontinuierlich aktualisiert wird. Leider hat in diesem Fall der asynchrone Reload der Komponenten nicht den gewünschten Effekt erzielt. Normalerweise sollte der Benutzer von dem regelmässigen Polling nichts merken, durch die Kombination von Polling und Progressbars kann das aber nicht vermieden werden. Einerseits aktualisiert sich jede Progressbar selbst in einem festgelegten Intervall, andererseits rendert die Polling Komponente alle Progressbars neu. Als Resultat erhält der Benutzer sich ständig neu aufbauende Progressbars. Eine Variante, um dieses Problem zu vermeiden, könnte folgendermaßen aussehen: Der Teacher öffnet eine Umfrage und wartet bis sich alle Studenten angemeldet haben. Danach schliesst er die Umfrage nach aussen hin, d.h. es dürfen nur diejenigen teilnehmen, die sich auch angemeldet haben. Diese Methode erlaubt es, die Liste aller Progressbars nur einmal aufzubauen, anschliessend aktualisiert jede den eigenen Zustand im festgelegten Intervall. Diese Variante funktioniert zwar problemlos, jedoch ähnelt der Aufbau vielmehr einem Test als einer Umfrage. Eine andere Möglichkeit dieses Problem zu umgehen wäre die Verwendung von JavaScript. Ein Poll befragt kontinuierlich die Logik, ob sich ein neuer Student für die Umfrage angemeldet hat. Trifft das zu, dann wird per JavaScript Aufruf eine weitere Progressbar freigeschalten. Auf diese Weise braucht kein Poll auf die Liste durchgeführt werden.

2.7.3 QTI Probleme

Eigentlich wurde der QTI playr des ASDEL Projekts nicht für die Verwendung innerhalb der JSF Umgebung ausgelegt, weshalb während der Integration drei grössere Probleme aufgetreten sind.

- Für die **Verbindung zwischen der Anwendung und dem playr**, wie sie in Abschnitt 2.3.2 beschrieben wurde, wird eine korrekte Installation und Konfiguration des playr vorausgesetzt (siehe Kapitel 3). Ist diese Hürde bewältigt, dann kann die Verbindung initialisiert werden, dabei müssen alle vorgegebenen Schritte systematisch eingehalten werden. Wird die Session ID nicht korrekt gespeichert oder das Content Package in einem falschen Format an den playr übermittelt, um nur einige der potentiellen Fehlerquellen anzuführen, dann wird die Kommunikation abgebrochen.
- Sobald der Verbindungsaufbau geglückt ist und dem Benutzer die erste Frage angezeigt wird, eröffnet sich auch schon das nächste Problem. Die Antwort des Benutzers wird in Form von **Parametern an den playr** gesandt, zuvor müssen diese Parameter aber korrekt aus der Seite extrahiert werden. In den meisten Fällen stellt das kein Problem dar, es sei denn bei dem Fragetyp werden Checkboxen mit multiplen Auswahlmöglichkeiten angeboten. Das Problem ergibt sich, sobald zwei oder mehr der auswählbaren Optionen selektiert und abgeschickt werden, denn jede dieser Optionen besitzt denselben Identifier, aber einen anderen Wert. Wird in der zuständigen Logik die Parameterliste mit dem falschen Aufruf geladen, dann wird nur eine der gewählten Antworten erfasst, da alle Checkboxen denselben Identifier benutzen und sich somit überlagern.
- Eine weitere Fehlerquelle betrifft die **JavaScript Funktionen** des playr. Diese Funktionen sollen die korrekte Anzeige der Fragen und Eingabe der Antworten garantieren. Für Checkboxen gibt es beispielsweise eine Funktion, die eine Meldung anzeigen soll, sobald der Benutzer mehr Antworten auswählt als in der Angabe gewünscht sind. Wie bereits in der Einleitung angemerkt wurde, ist der playr nicht für die JSF Umgebung ausgelegt, weshalb auch einige der JavaScript Funktionen nicht das gewünschte Resultat liefern. Im Gegensatz zu JavaServer Pages übermittelt JSF in Kombination mit den Komponentenbibliotheken nicht nur das Formular, in dem die grafische Oberfläche des playr eingebettet ist, sondern auch versteckte (hidden) Formulare. Die JavaScript Funktionen wurden aber so implementiert,

dass sie über den Index auf alle Komponenten zugreifen. Bei den Formularen kann das unter Umständen nicht zum gewünschten Ergebnis führen, da das erwartete Formular mit dem QTI playr nicht mehr an erster Stelle stehen muss. Um das Problem zu beheben mussten alle JavaScript Funktionen umgeschrieben werden, damit der Zugriff auf die Komponenten über einen Identifier funktioniert und nicht wie vorher über den Index.

- Der vierte und letzte Punkte behandelt das "HTML Munging" (siehe Abbildung 2.5 in Abschnitt 2.3.2). Nachdem der playr die nächste anzuzeigende Frage im HTML Format an die Anwendung gesendet hat, müssen noch einige Adaptionen im Code durchgeführt werden, damit die Fragen korrekt dargestellt wird. Einerseits müssen die Pfade für die verwendeten JavaScript Dateien und Bilder angepasst werden, andererseits müssen alle "asdel.jsp" Einträge durch den Namen der aktuellen Seite ersetzt werden, damit alle Formulare richtig abgeschickt werden.

Kapitel 3

Installations- und Bedienungsanleitung

Dieses Kapitel ist in zwei Teile untergliedert, die Installations- und die Bedienungsanleitung. Die Installationsanleitung beschreibt alle nötigen Schritte um die Anwendung starten zu können, angefangen beim Einrichten der Datenbank bis hin zum Einbinden des playr. Der zweite Teil des Kapitels, die Bedienungsanleitung, soll den Einstieg und die Handhabung der Applikation erleichtern.

3.1 Installation

Die Installation läuft in mehreren Schritten ab, zuerst werden die Datenbank und Tomcat eingerichtet, anschliessend der playr konfiguriert, und schliesslich die Anwendung selbst installiert. Bevor jedoch mit der Installation begonnen wird sollte überprüft werden, ob die folgenden Voraussetzungen erfüllt sind:

1. Eine aktuelle Java Runtime Environment (JRE) in der Version 1.6.* muss am Zielrechner installiert sein.
2. Für die Betrachtung der Webanwendung sollte einer der folgenden Browser in der angegebenen oder einer aktuelleren Version verwendet werden:

- Mozilla Firefox 3.0.5
- Microsoft Internet Explorer 7.0.5730.11
- Google Chrome 1.0.154.43
- Apple Safari 3.2.1

Alle Dateien, die für die Installation notwendig sind, befinden sich auf einer der Arbeit beigelegten CD.

3.1.1 Datenbank

Folgende Schritte müssen eingehalten werden, um die Datenbank passend für die Anwendung einzurichten:

1. Aktuelle Version von MySQL und MySQL Tools von der Seite <http://dev.mysql.com/downloads/> herunterladen
2. MySQL Installationsmenü starten und den Anweisungen folgen (die vorgeschlagenen Einstellungen können alle beibehalten werden)
3. Als Port *3010* auswählen
4. Username ist *root* und Passwort *test* (falls diese Einstellungen in MySQL geändert werden, muss die Hibernate Konfigurationsdatei *hibernate.cfg.xml* im classes Ordner angepasst werden)
5. Installation beenden und Datenbank starten
6. Neue Datenbank mit dem Namen *education* erzeugen - dazu den MySQL Query Browser starten und im Reiter Schemata per rechten Mausklick ein neues Schema anlegen bzw. in der MySQL Eingabeaufforderung den Befehl `CREATE DATABASE education;` eingeben und ausführen

3.1.2 Tomcat

Die Installation von Apache Tomcat verläuft verhältnismässig einfach. Es muss dazu nur eine aktuelle Version (6.0.*) heruntergeladen und in den Programm-Ordner entpackt werden. Anschliessend sollte überprüft werden, dass sich keine Leerzeichen im Namen des neuen Tomcat-Ordners befinden, da dies unter verschiedenen Systemen zu

Problemen führen kann. Tomcat verwendet standardmäßig den Port 8080, sollte dieser bereits belegt sein, dann kann der Port in der Datei `Server.xml` im Tomcat Verzeichnis entsprechend angepasst werden. Die exakte Pfadangabe der Tomcat Installation wird zu einem späteren Zeitpunkt für die Konfiguration nochmals benötigt.

3.1.3 QTI playr

Bevor der QTI playr selbst integriert werden kann, muss das Framework R2Q2 eingebunden werden. Dazu werden folgende Schritte benötigt:

1. Die Datei `r2q2.war` muss von der Installations-CD in den Ordner `webapps` im Tomcat Verzeichnis kopiert werden.
2. Tomcat neu starten
3. Webbrowser öffnen und `http://localhost:8080/r2q2/r2q2install.jsp` aufrufen
4. Den "Start Script" Knopf betätigen (generiert automatisch die Datei `r2q2.properties`)
5. Wenn eine Bestätigungs-Meldung angezeigt wird, kann der Browser geschlossen werden

Nun kann der playr installiert werden:

1. Den Ordner `asdel` von der Installations-CD in das Wurzelverzeichnis des Tomcat kopieren (enthält alle Konfigurationsdateien)
2. Die Datei `playr.war` von der Installations-CD in den Ordner `webapps` im Tomcat Verzeichnis kopieren

3.1.4 Anwendung

Ist soweit alles eingerichtet, muss nur mehr die Datei `platform.war` von der Installations-CD in den Ordner `webapps` im Tomcat Verzeichnis kopiert werden. Sobald Tomcat hochgefahren wird, erscheint für jede Datei mit der Endung `.war` im Ordner `webapps` ein neuer Ordner (der Ordner `r2q2` sollte an dieser Stelle bereits auf Grund von Punkt 3.1.3 existieren). Abschliessend können noch einige Konfigurationsdateien angepasst werden:

- **hibernate.cfg.xml**: über diese Datei lässt sich die Datenbankanbindung konfigurieren (Datenbank, Port, Username, Passwort, Schema)
- **log4j.properties**: der Eintrag *log4j.appender.R.File* definiert die Log-Datei
- **configuration.properties**: über die angeführten Pfade werden die Verzeichnisse der Unterordner (upload, reports, templates) bestimmt und *playrUrl* beschreibt die URL des playr (der Pfad sollte allerdings im Normalfall nicht geändert werden)

Nachdem alle benötigten Frameworks integriert und konfiguriert wurden, kann die Anwendung aufgerufen werden. Im Normalfall erreicht man die Login Seite über den folgenden Link:

<http://localhost:8080/diplom>

3.2 Bedienungsanleitung

Um neuen Benutzern den Einstieg zu vereinfachen, werden in diesem Kapitel die wichtigsten Anwendungsszenarien anhand von Abbildungen beschrieben. Der Login und Logout wird nur im ersten Use Case erwähnt, da er sich nicht ändert.

3.2.1 Use Case 1 - Umfrage manuell oder auf Basis eines QTI Content Package erstellen

- Über den Login kann sich ein Benutzer am System anmelden (Abbildung 3.1)

Login Page

Name	<input type="text" value="Test_Teacher"/>
Password	<input type="password" value="...."/>
	<input type="button" value="Login"/>

Abbildung 3.1: Login

- Auf der Umfragen Übersicht können alle bisher erstellten Umfragen betrachtet, gelöscht oder geöffnet werden (Abbildung 3.2)

Search Quiz: [+](#) **QTI**

ID	Name	Created by	Creation Date	Status	Type	Actions
13	Final Test	Test_Teacher	2008-09-07 12:53:14.0	INITIALIZED	COMPOSED	
15	Another test	Test_Teacher	2008-09-19 08:04:44.0	INITIALIZED	COMPOSED	
16	Normal Test	Test_Teacher	2008-09-21 10:45:04.0	INITIALIZED	COMPOSED	
20	QTI Test	Test_Teacher	2008-10-02 11:23:40.0	INITIALIZED	QTI	
21	Test with optional line	Test_Teacher	2008-10-03 09:34:27.0	INITIALIZED	COMPOSED	
23	Test with optional line 2	Test_Teacher	2008-10-03 10:39:41.0	INITIALIZED	COMPOSED	
24	New Radio Test	Test_Teacher	2008-10-03 11:09:54.0	INITIALIZED	COMPOSED	
25	A new test	Test_Teacher	2008-10-21 12:20:47.0	INITIALIZED	QTI	
27	Multiple Choices	Test_Teacher	2008-12-30 18:47:59.0	INITIALIZED	COMPOSED	
28	Rating Test	Test_Teacher	2009-01-03 12:16:52.0	INITIALIZED	COMPOSED	

Abbildung 3.2: Umfragen Übersicht

- Über das + Symbol auf der Umfragen Übersicht erreicht man die Seite für die Erstellung manueller Umfragen. Abbildungen 3.3 bis 3.7 zeigen die nötigen Schritte, um eine neue Umfrage anzulegen

In Schritt 1 wird der Name der Umfrage festgelegt

Quiz Name

Name: [Create Quiz](#)

Abbildung 3.3: Schritt 1 - Name der Umfrage eingeben

In Schritt 2 kann eine neue Frage hinzugefügt werden

The screenshot shows two main sections. The top section, titled 'Quiz Name', contains a text input field with the value 'Neue Umfrage' and a 'Create Quiz' button to its right. The bottom section, titled 'Questions', contains a green '+ Add new question' button. Below these sections is a 'Cancel Quiz' button.

Abbildung 3.4: Schritt 2 - Frage hinzufügen

In Schritt 3 wird für die neue Frage der Text, der Fragetyp und die Anzahl der möglichen Antworten definiert

The screenshot shows the 'Question Settings' section. It includes a 'Question:' text input field with the text 'Bitte geben Sie Ihren Namen ein'. Below it is a 'Type:' dropdown menu set to 'Single line input'. At the bottom of this section is a 'Possible answers:' spinner control set to '1'. Below the settings are 'Confirm' and 'Cancel' buttons.

Abbildung 3.5: Schritt 3 - Frage festlegen

In Schritt 4 wird für jede mögliche Antwort ein Label angegeben

The screenshot shows the 'Your question is:' section with a text input field containing 'Bitte geben Sie Ihren Namen ein'. Below it is the 'Please fill out the labels for the possible answers' section, which contains a 'Name' label and an empty text input field. At the bottom is a 'Finish Question' button.

Abbildung 3.6: Schritt 4 - Label definieren

In Schritt 5 kann nun entweder die Umfrage gespeichert oder weitere Fragen hinzugefügt werden

Quiz Name

Name: [Create Quiz](#)

Questions

[Add new question +](#)

Bitte geben Sie Ihren Namen ein ✘

Abbildung 3.7: Schritt 5 - Umfrage speichern oder neue Frage hinzufügen

- Über das QTI Symbol auf der Umfragen Übersicht erreicht man die Seite, auf der ein QTI Content Package hochgeladen und daraus eine Umfrage generiert werden kann (Abbildung 3.8)

QTI Details

Quizname:

/Users/michael/Uni/workspace_diplom/platform/upload/Arc Done	Clear
--	-----------------------

Abbildung 3.8: Laden einer QTI Umfrage

- Um sich vom System abzumelden braucht man nur den Logout Button im oberen rechten Eck zu betätigen (Abbildung 3.9)

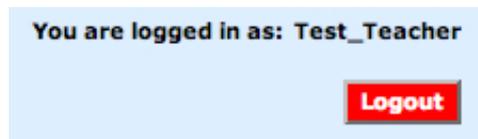


Abbildung 3.9: Logout

3.2.2 Use Case 2 - Durchführung einer Umfrage aus der Sicht eines Lehrers

- Über das Bleistift Symbol auf der Umfragen Übersicht erreicht man die Seite für die Durchführung von manuellen und QTI Umfragen. Abbildungen 3.10 bis 3.12 zeigen die nötigen Schritte, um eine Umfrage abzuhalten.

In Schritt 1 müssen der ausgewählten Umfrage eine oder mehrere Usergruppen zugewiesen werden, wahlweise kann auch der Eintrag VISITOR für eine anonyme Umfrage selektiert werden. Anschliessend muss über den Link 'Click to set status to OPEN' die Umfrage freigeschaltet werden.

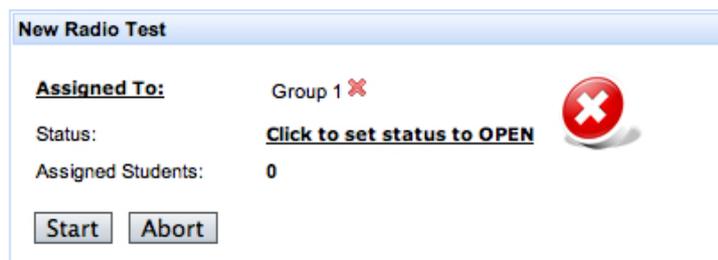


Abbildung 3.10: Schritt 1 - Umfrage zuweisen

In Schritt 2 können sich Studenten für die Umfrage anmelden, dabei wird die aktuelle Anzahl der Teilnehmer unter dem Punkt 'Assigned Students' angeführt. Anschliessend wird per Druck auf den Start Knopf die Umfrage begonnen.



Abbildung 3.11: Schritt 2 - Studenten melden sich an

In Schritt 3 kann der Lehrer den Fortschritt jedes einzelnen Studenten mitverfolgen, und wenn nötig die gesamte oder auch nur eine individuelle Umfrage abbrechen.

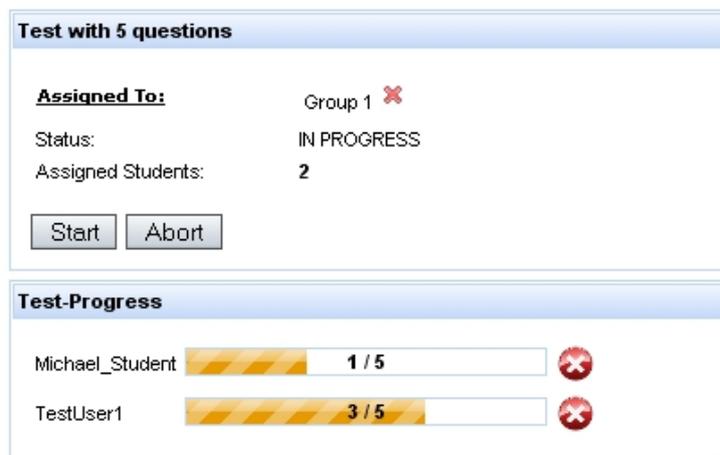


Abbildung 3.12: Schritt 3 - Fortschritt der Umfrage beobachten

3.2.3 Use Case 3 - Teilnahme an einer Umfrage aus der Sicht eines Studenten

- Die Abbildungen 3.13 bis 3.17 zeigen die Vorgehensweise bei einer Umfrage aus der Sicht eines Studenten.

In Schritt 1 wählt der Student den Menüpunkt 'Quizzes / Tests' in der Sidebar, muss jedoch feststellen, dass noch keine Umfrage freigeschaltet ist.

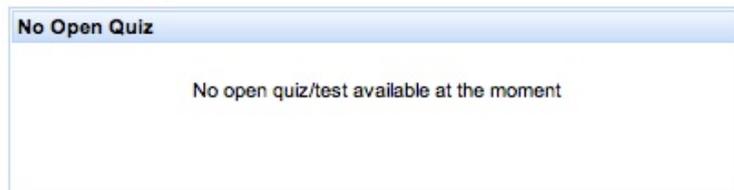


Abbildung 3.13: Schritt 1 - Auf Umfrage warten

In Schritt 2 ändert sich die Anzeige und die Beschreibung der offenen Umfrage erscheint. Per Klick auf den 'Take Test' Button wird die Anmeldung an der Umfrage durchgeführt.



Abbildung 3.14: Schritt 2 - Bei Umfrage anmelden

In Schritt 3 wird der Student automatisch auf eine neue Webseite weitergeleitet und kann dort die Umfrage starten.

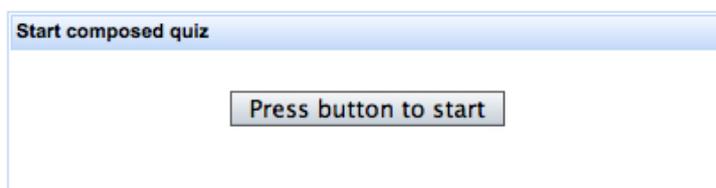


Abbildung 3.15: Schritt 3 - Umfrage starten

In Schritt 4 kann der Student alle selbst erstellten Fragen beantworten und wird nach dem Beenden der Umfrage wieder auf die Ausgangsseite weitergeleitet.

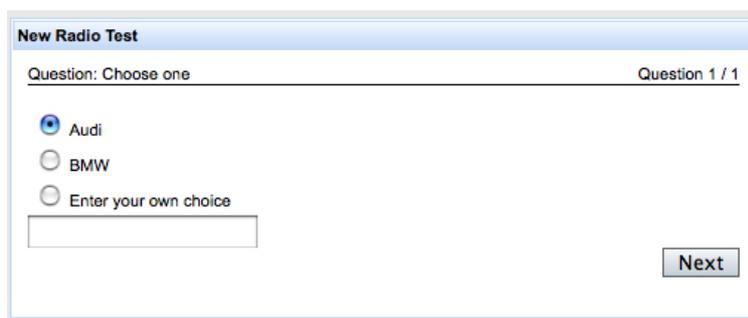


Abbildung 3.16: Schritt 4 - Selbst erstellte Umfrage beantworten

Falls es sich um eine QTI Umfrage handelt, sieht die Darstellung etwas anders aus.

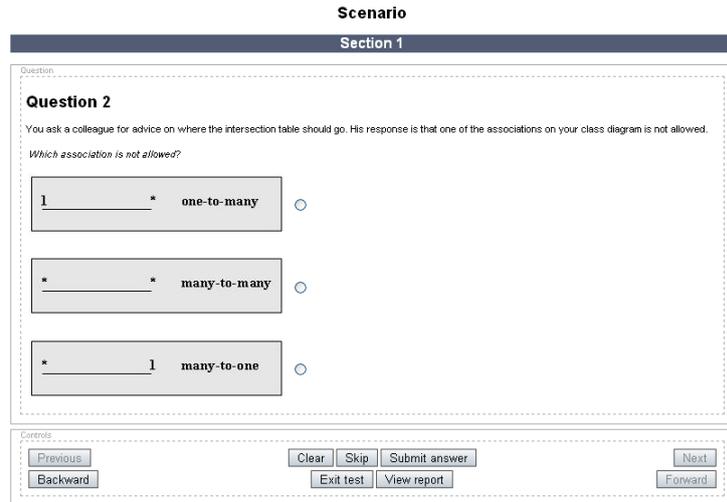


Abbildung 3.17: Schritt 4 - QTI Umfrage beantworten

3.2.4 Use Case 4 - Teilnahme an einer anonymen Umfrage aus der Sicht eines Studenten

- Wie bereits in Use Case 3 zeigen die Abbildungen 3.18 bis 3.22 die Vorgehensweise bei einer Umfrage aus der Sicht eines Studenten, mit dem grossen Unterschied, dass nun alle Teilnehmer anonym am System angemeldet sind, d.h. es findet kein expliziter Login statt, sondern die Studenten erreichen über einen definierten Link sofort die Webplattform.

In Schritt 1 folgt der Student dem Link für die anonyme Anmeldung und wird auf die Umfragen-Seite weitergeleitet, muss jedoch feststellen, dass noch keine Umfrage freigeschaltet ist.

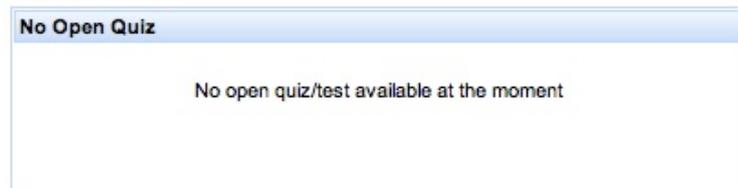


Abbildung 3.18: Schritt 1 - Auf Umfrage warten

In Schritt 2 ändert sich die Anzeige und die Beschreibung der offenen Umfrage erscheint. Per Klick auf den 'Take Test' Button wird die anonyme Anmeldung an der Umfrage durchgeführt.

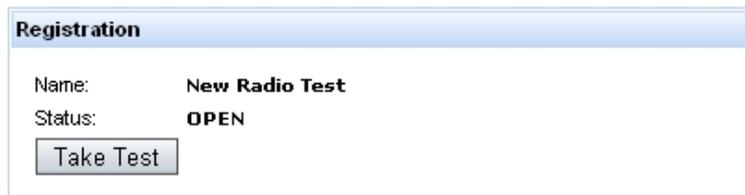


Abbildung 3.19: Schritt 2 - Bei Umfrage anmelden

In Schritt 3 wird der Student automatisch auf eine neue Webseite weitergeleitet und kann dort die Umfrage starten.

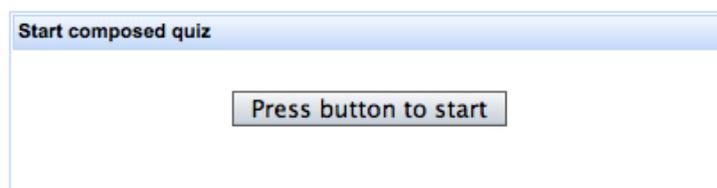


Abbildung 3.20: Schritt 3 - Umfrage starten

In Schritt 4 kann der Student alle selbst erstellten Fragen beantworten und wird nach dem Beenden der Umfrage wieder auf die Ausgangsseite weitergeleitet.

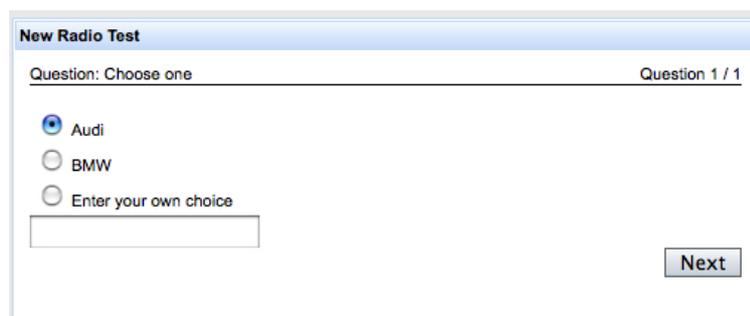


Abbildung 3.21: Schritt 4 - Selbst erstellte Umfrage beantworten

Falls es sich um eine QTI Umfrage handelt, sieht die Darstellung etwas anders aus.

Scenario
Section 1

Question

Question 2
You ask a colleague for advice on where the intersection table should go. His response is that one of the associations on your class diagram is not allowed.
Which association is not allowed?

1 _____ * one-to-many

* _____ * many-to-many

* _____ 1 many-to-one

Controls

Previous Clear Skip Submit answer Next
Backward Exit test View report Forward

Abbildung 3.22: Schritt 4 - QTI Umfrage beantworten

3.2.5 Use Case 5 - Benutzeradministration

- Über die Benutzer Übersicht können autorisierte Personen Benutzer und Benutzergruppen anlegen, löschen und zuweisen (Abbildung 3.23)

User Overview

Search User:

User ID	User Name	User Role	Assigned Groups	Actions
Anonym-673287276	Anonym	VISITOR	Group 1	<input type="checkbox"/> <input type="checkbox"/>
dummy-1128223867	dummy	ADMIN		<input type="checkbox"/> <input type="checkbox"/>
Michael_Student1900003567	Michael_Student	STUDENT	Group 1	<input type="checkbox"/> <input type="checkbox"/>
TestUser11466019966	TestUser1	STUDENT	Group 1	<input type="checkbox"/> <input type="checkbox"/>
TestUser2-1749986873	TestUser2	STUDENT	Group 1	<input type="checkbox"/> <input type="checkbox"/>
Test_Admin16734252	Test_Admin	ADMIN		<input type="checkbox"/> <input type="checkbox"/>
Test_Teacher135276138	Test_Teacher	ADMIN		<input type="checkbox"/> <input type="checkbox"/>

Abbildung 3.23: Benutzer Übersicht

Kapitel 4

Erweiterungsmöglichkeiten

Die Anwendung ist in ihrem derzeitigen Zustand funktionell sehr spezialisiert. Wie aus der Arbeit hervorgeht besteht die Hauptfunktionalität in der Durchführung von Umfragen und ist für die Verwendung im Unterricht ausgelegt. Auf diese Weise wird dem Vortragenden die Möglichkeit geboten, während des Unterrichts Fragen an alle Teilnehmer gleichzeitig zu richten und auszuwerten und somit die Lehreinheiten interaktiver zu gestalten.

Im Hinblick auf andere Lernplattformen wie Sakai oder Moodle würden sich zahlreiche Erweiterungsmöglichkeiten anbieten. An dieser Stelle soll aber keine Auflistung von möglichen neuen Funktionen folgen, sondern einige Punkte behandelt werden, wie sich das bestehende Umfrage System noch erweitern ließe. In der nachstehenden Liste werden fünf interessante Ideen präsentiert:

- Während einer Umfrage könnte sich ein **Chat**, der die Echtzeit-Kommunikation zwischen Teilnehmer und Vortragenden zulässt, als hilfreich erweisen. Das ermöglicht den Teilnehmern Fragen an den Vortragenden zu richten, falls etwas unklar sein sollte. Besonders bei anonymen Umfragen würde sich ein (ebenfalls anonymer) Chat anbieten, da auf diese Weise der Vortragende nicht erfährt, wer die Frage verfasst hat. Auch sind die Teilnehmer oftmals gehemmt, Fragen in der Gegenwart Anderer zu stellen, weshalb ein Chat das Fragen angenehmer gestalten würde.
- Momentan ist das System für die Durchführung von Umfragen ausgelegt, es kann aber auch ohne Weiteres für **Tests** herangezogen werden. In letzterem Fall könnte die Auswertung eines Tests für den Vortragenden etwas vereinfacht werden, indem der Vortragende beim Erstellen eines Tests zugleich auch die korrekten Antworten

ten definiert. Nachdem alle Teilnehmer ihre Tests absolviert haben, übernimmt ein automatisierter Prozess die Korrektur. Diese Methode lässt sich für Multiple-Choice Tests problemlos einsetzen, bei textuellen Eingaben gestaltet sich die Autokorrektur bereits etwas schwieriger. Hier könnte die Antwort beispielsweise auf Schlüsselwörter durchsucht und dementsprechend bewertet werden.

- Nachdem eine Umfrage beendet ist, kann der Vortragende die **Resultate** begutachten. In der derzeitigen Implementierung steht als Ausgabeformat nur HTML zur Verfügung. Im Rahmen einer Erweiterung könnte beispielsweise ein PDF oder XLS Generator integriert werden, der die Ergebnisse in das gewünschte Format umwandelt und ausgibt.
- Die aktuelle Version der Anwendung ist auf das Anzeigen von QTI Content Packages durch den *player* begrenzt. Mittlerweile stellt das ASDEL Projekt **weitere Frameworks** wie den *constructr* oder *assessr* zur Verfügung (siehe Abschnitt 2.3.1), die in die Applikation aufgenommen werden könnten. Der *constructr* ermöglicht das Erstellen von QTI Fragen während der *assessr* das Auswerten derselben übernimmt.
- Hinsichtlich dem vorhergehenden Punkt würde sich eine *QTI Datenbank* anbieten, um selbst erstellte QTI Fragen und Content Packages warten und Anderen öffentlich zugänglich machen zu können. Die University of Southampton bietet zum Beispiel unter dem Namen *Minibix* eine *QTI Item Bank* an.

Diese Vorschläge sollen nur als Beispiele dienen, es bleibt jedem Benutzer selbst überlassen, in welche Richtung die bestehende Anwendung erweitert werden soll.

Literaturverzeichnis

- [ASDEL] Assessment Delivery Engine, <http://www.asdel.ecs.soton.ac.uk/>, 20.11.2008
- [ASDEL1] ASDEL - assessment delivery engine for QTiv2 questions, <http://www.asdel.ecs.soton.ac.uk/presentations/ASDEL-5-6-07.pdf>, 20.11.2008
- [ASDEL2] Gary Wills, Hugh Davis, Lester Gilbert, Jonathon Hare, Yvonne Howard, Steve Jeyes, David Millard, and Robert Sherratt. Delivery of QTiv2 Question Types. Learning Societies Lab, University of Southampton, UK
- [HIB] Hibernate - Relational Persistence for Java and .NET, <http://www.hibernate.org/>, 20.11.2008
- [HIB08] Richard Oates, Thomas Langer, Stefan Wille, Torsten Lueckow, Gerald Bachlmayr. Spring & Hibernate - Eine praxisbezogene Einführung. Seite 6. 2008, München / Wien, Carl Hanser Verlag
- [HIBPER] Hiberante - Persistence Lifecycle, <http://scbcd.blogspot.com/2007/01/hibernate-persistence-lifecycle.html>, 20.11.2008
- [ILIAS] Integriertes Lern-, Informations- und Arbeitskooperations-System (ILIAS), <http://www.ilias.de/>, 20.11.2008
- [IMS] IMS Global Consortium, <http://www.imsglobal.org/question/>, 20.11.2008
- [IMS_SPEC] IMS Question and Test Interoperability Assessment Test, Section, and Item Information Model - Version 2.1 Public Draft revision 2 Specification, http://www.imsglobal.org/question/qtiv2p1pd2/imsqti_infov2p1pd2.html, 20.11.2008

[IMS_TYPES] , http://www.imsglobal.org/question/qti_v2p0/imsqti_implv2p0.html,
08.01.2009

[JSF] The Life Cycle of a JavaServer Faces Page,
<http://java.sun.com/javaee/5/docs/tutorial/doc/bnaqq.html>,20.11.2008

[JSF06] Bernd Müller. Java Server Faces - Ein Arbeitsbuch für die Praxis. Seite 49-54.
2006, München / Wien, Carl Hanser Verlag

[MOODLE] Modular Object-Oriented Dynamic Learning Environment (Moodle),
<http://moodle.org/>, 20.11.2008

[OLAT] Online Learning And Training, <http://www.olat.org/website/en/html/index.html>,
20.11.2008

[QM] QuestionMark, <http://www.questionmark.com/deu/index.aspx>, 20.11.2008

[QTITOOLS] QTITools, http://wiki.qtitools.org/wiki/Main_Page, 20.11.2008

[SAKAI] Sakai, <http://www.sakaiproject.org/portal>, 20.11.2008

Curriculum Vitae

Persönliche Daten

<i>Name</i>	Michael Prinz
<i>Akademischer Grad</i>	Bakk. techn.
<i>Geburtsdatum</i>	23.Juli 1982
<i>Geburtsort</i>	Linz
<i>Wohnort</i>	4209 Haid
<i>Strasse</i>	Haidberg 1
<i>E-Mail Adresse</i>	Prinz.Michael@gmx.at
<i>Familienstand</i>	ledig

Ausbildung

<i>2001 -</i>	Masterstudium Informatik an der Johannes Kepler Universität
<i>1992 - 2000</i>	Bischöfliches Gymnasium Petrinum

Berufserfahrung

<i>April 2006 -</i>	Freier Dienstnehmer bei Webdynamite IT Solutions als Softwareentwickler im Bereich Java J2EE Webapplikationen
<i>Okt. 2006 - Juni 2008</i>	Voestalpine Eurostahl CMS Unterstützung
<i>2006 - 2007</i>	Tutor für die Lehrveranstaltung Softwareentwicklung 2 am Institut für Telekooperation an der Johannes Kepler Universität
<i>November 2005 -</i>	Barkeeper für Bellini Cocktailservice
<i>Sept. 2002 - Juni 2007</i>	Freier Dienstnehmer bei Sixt - Zuständig für die Zustellung und Wartung von Fahrzeugen
<i>Sept. 2000 - Dez. 2000</i>	Angestelltenverhältnis bei Karrer und Partner im Bereich der Kundenbetreuung

Sprachkenntnisse

Englisch fliegend in Schrift und Sprache
Französisch Grundkenntnisse

Fachkenntnisse

Formale Sprachen Java(script), C++, C#, Pascal, php, HTML, SQL, CSS, XML
Systeme Microsoft Windows, Mac OS X
Frameworks/Bibliotheken Hibernate, Spring, Servlet, JSP, JSF (MyFaces, Tomahawk, Trinidad, RichFaces), Struts
Entwicklungsumgebungen (My)Eclipse, Microsoft Visual Studio
Software Microsoft Office, Open Office, Apache Tomcat, MySQL, Oracle Database, SQL Developer, Toad, CVS, SVN, JMeter

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Diplom- bzw. Masterarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe.

Ort und Datum

Michael Prinz